SIP Server Platform

Reference Guide

RADVISION

Delivering the Visual Experience

NOTICE

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

RADVISION SIP Server Platform version 3.5, March 2008

Publication 10

http://www.radvision.com

CONTENTS

REGISTER SERVER MODULE

3 *Register Server Functions*

SERVER AUTHENTICATION MODULE

4 *Server Authentication Functions*

TRANSPORT MODULE

COMPONENTS MODULE

SIP SERVER LIST MODULE

SIP SERVER STD LAYER

8 *SIP Server STD Functions*

MEMORY POOL MODULE

MIDDLE LAYER MODULE

TYPE DEFINITIONS AND STATUS CODES

About this Reference Guide

The RADVISION SIP Server Platform is a comprehensive software framework for development of SIP server applications. The Platform includes an implementation of fully-standard server functionality that is controllable through multi-level Application Programming Interfaces (APIs). SIP Server developers can quickly and effectively add SIP Server capabilities to their application by integrating the SIP Server library.

The *RADVISION SIP Server Platform Reference Guide* and the *RADVISION SIP Server Platform Message Layer Reference Guide* are for developers of RADVISION SIP Server applications. These Reference Guides list the Application Programming Interfaces (API) functions of the SIP Server Platform. Each function has a general description, syntax description and other important information. The Reference Guides also contain the different enumerations and structure types that are defined in the API.

Documentation

See the *SIP Server Platform Release Notes* for a complete list of documentation for the SIP Server Platform and SIP Server Platform Add-on Modules.

ADDITIONAL PRODUCTS

The SIP Server Platform is part of the RADVISION SIP product family which also includes these products:

RADVISION IMS SIP TOOLKIT (STACK)

The RADVISION IMS SIP Toolkit consists of SIP, SDP and RTP/RTCP Stacks, a suite of APIs, sample code and user documentation for developers who wish to build SIP applications. The Toolkit includes IMS support. For more information, see the *RADVISION SIP Stack Programmer Guide* and the *RADVISION IMS SIP Add-on Module Programmer and Reference Guide*.

PROLAB™ SIP TEST MANAGER

The ProLab™ SIP Test Manager is a highly scalable, feature-rich testing system that is suitable for use in different stages of the product development cycle. The ProLab™ SIP Test Manager can emulate a group of synchronized SIP entities, such as SIP user agents (UAs) and SIP Servers. For more information, see the *RADVISION ProLab™ SIP Test Manager User Guide.*

MULTI-MEDIA TERMINAL FRAMEWORK

The RADVISION Multi-media Terminal Framework provides VoIP application developers with an extensible framework for developing IP-based videophones, IP phones and IADs. It shields the application from the myriad tasks that a terminal must implement. It also enables application developers to concentrate on value-added services rather than on the intricacies of VoIP signaling, security, call state handling, and standard supplementary services. A complete terminal solution can be achieved by integrating the RADVISION Multi-media Terminal Framework components with third-party DSP services and telephony hardware. For more information, see the *RADVISION Multi-media Terminal Framework Programmer Guide*.

SIP Server Manager Module

The SIP Server Manager (*SipServerMgr)* contains SIP Server Manager API functions that enable you to initialize, construct and destruct the SIP Server. These functions also enable you to control logging and logging levels during runtime, obtain the resource status, and set the application implementations for the Server Components Interface functions.

This part includes the following section:

- SIP Server Manager Functions

1

SIP SERVER MANAGER FUNCTIONS

WHAT'S IN THIS SECTION

This section contains the SIP Server Manager API functions included in the *RvSipServerManager.h* header file.

The SIP Server Manager API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvSipServerMgrConstruct()
- RvSipServerMgrDestruct()
- RvSipServerMgrInitCfg()
- RvSipServerMgrProcessEvents()
- RvSipServerMgrSelect()
- RvSipServerMgrSelectUntil()
- RvSipServerMgrConstructFromFile()
- RvSipServerMgrConstructB2BAFAndB2BAFServiceLib()
- RvSipServerMgrInitB2BAFAndB2BAFServiceLibCfg()
- RvSipServerMgrConstructB2BAFAndB2BAFServiceLibFromFile()
- RvSipServerMgrConstructActModule()
- RvSipServerMgrDestructActModule()

RvSipServerMgrConstruct()

DESCRIPTION

Constructs and initializes a SIP Server instance. This function allocates the required memory and constructs the SIP Server objects according to configuration parameters specified in the RvSipServerCfg structure. The SIP Stack is also constructed according to configuration parameters. This function returns a handle to the *SipServerMgr*. You need this handle in order to use the SIP Server Platform API functions.

SYNTAX

```
RvStatus RvSipServerMgrConstruct(
    IN    RvInt32                 sizeofCfg,
    INOUT RvSipServerCfg*         pServerCfg,
    OUT   RvSipServerMgrHandle*   hSipServerMgr);
```

PARAMETERS

pServerCfg

A structure containing the SIP Server and the SIP Stack configuration parameters. This parameter is also an output parameter. The SIP Server gives default values to fields that contain invalid values and have not been initialized.

sizeOfCfg

The size of the configuration structure.

hSipServerMgr

The handle to the *SipServerMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrDestruct()

DESCRIPTION

Destructs and terminates the SIP Server instance. This function destroys all SIP Server modules and frees all the memory that was allocated. The SIP Stack Destruct function is called.

SYNTAX

```
RvStatus RvSipServerMgrDestruct(
    IN RvSipServerMgrHandle    hSipServerMgr);
```

PARAMETERS

hSipServerMgr

The handle to the SIP Server instance.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrInitCfg()

DESCRIPTION

Initializes the configuration parameters specified in the RvSipServerCfg structure with the SIP Server default values. You should use this function before calling RvSipServerMgrConstruct() to initialize the configuration structure.

SYNTAX

```
void RvSipServerMgrInitCfg(
    IN  RvUint32           sizeofCfg,
    OUT RvSipServerCfg*    pServerCfg);
```

PARAMETERS

sizeOfCfg

The size of the configuration structure.

pServerCfg

The configuration structure containing the SIP Server default values.

RETURN VALUES

None.

RvSipServerMgrProcessEvents()

DESCRIPTION

Calls the RvSipStackProcessEvents() function that checks for events and processes them as they come. This function is usually used for console applications.

SYNTAX

```
void RvSipServerMgrProcessEvents();
```

PARAMETERS

None.

RETURN VALUES

None.

RvSipServerMgrSelect()

DESCRIPTION

Performs one call to the Select() function on operating systems that support the select interface. An application should write the following as its main loop:

```
while (1) RvSipServerMgrSelect();
```

SYNTAX

```
RvStatus RvSipServerMgrSelect();
```

PARAMETERS

None.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrSelectUntil()

DESCRIPTION

On operating systems that support the select interface, this function will perform one call to the Select() function. This function also gives the application the ability to give a maximum blocking delay.

SYNTAX

```
RvStatus RvSipServerMgrSelectUntil(
    IN RvUint32    delay);
```

PARAMETERS

delay

The maximum time to block in milliseconds.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrConstructFromFile()

DESCRIPTION

Constructs and initializes the SIP Server Stack. This function allocates the required memory and constructs SIP Server Stack objects according to the configuration parameters specified in the szCfgFilename file. The SIP Stack is also constructed according to configuration parameters. This function returns a handle to the SipServerMgr. You need this handle to use the SIP Server Stack API functions.

SYNTAX

```
RvStatus RvSipServerMgrConstructFromFile(
    IN RvChar*                      szCfgFilename,
    IN RvSipServerMgrEvHandler*     pEvHandler,
    OUT RvSipServerMgrHandle*       hSipServerMgr);
```

PARAMETERS

szCfgFilename

The XML filename containing SIP Server configuration.

pEvHandler

A pointer to the SipServerMgr event handler.

hSipServerMgr

The handle to the SipServerMgr.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrConstructB2BAFAndB2BAFServiceLib()

DESCRIPTION

Instructs the *SIPServerManager* to construct the B2BAF and B2BAF Service Library modules using *pB2BAFCfg* and *pB2BAFServiceLibCfg*. This function returns a handle to the created *B2BAFMgr* and the B2BAF *ServerManager*.

SYNTAX

```
RvStatus RvSipServerMgrConstructB2BAFAndB2BAFServiceLib(
  IN  RvSipServerMgrHandle            hSipServerMgr,
  IN  RvSipServerB2BAFCfg*            pB2BAFCfg,
  IN  RvSipServerB2BAFServiceLibCfg*  pB2BAFServiceLibCfg,
  OUT RvSipServerB2BAFMgrHandle*      phB2BAFMgr,
  OUT RvSipServerB2BAFServiceLibMgrHandle*
                                      phB2BAFServiceLibMgr);
```

PARAMETERS

hSipServerMgr

The handle to the *SipServerMgr*.

pB2BAFCfg

The B2BAF module configuration.

pB2BAFServiceLibCfg

The B2BAF Service Library module configuration.

phB2BAFMgr

The returned handle to the created *B2BAFMgr*.

phB2BAFServiceLibMgr

The returned handle to the created B2BAF *ServiceLibMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrInitB2BAFAndB2BAFServiceLibCfg()

DESCRIPTION

Initializes the *pB2BAFCfg* and *pB2BAFServiceLibCfg* with default values

SYNTAX

```
RvStatus RvSipServerMgrInitB2BAFAndB2BAFServiceLibCfg(
   IN RvSipServerB2BAFCfg*             pB2BAFCfg,
   IN RvSipServerB2BAFServiceLibCfg*   pB2BAFServiceLibCfg);
```

PARAMETERS

pB2BAFCfg

The configuration structure of the B2BAF module.

pB2BAFServiceLibCfg

The configuration structure of the B2BAF Service Library module.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrConstructB2BAFAndB2BAFServiceLibFromFile()

DESCRIPTION

Instructs the *SipServerMgr* to construct and initialize the B2BAF and B2BAF Service Library modules based on the configuration parameters specified in the *szCfgFilename* file.

SYNTAX

```
RvStatus
RvSipServerMgrConstructB2BAFAndB2BAFServiceLibFromFile(
    IN  RvSipServerMgrHandle          hSipServerMgr,
    IN  RvChar*                       szCfgFilename,
    OUT RvSipServerB2BAFMgrHandle*    phB2BAFMgr,
    OUT RvSipServerB2BAFServiceLibMgrHandle*
                                      phB2BAFServiceLibMgr);
```

PARAMETERS

hSipServerMgr

The handle to the *SipServerMgr*.

szCfgFilename

The XML filename containing SIP Server configuration.

phB2BAFMgr

The returned handle to the created *B2BAFMgr*.

phB2BAFServiceLibMgr

The returned handle to the created B2BAF *ServiceLibMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrConstructActModule()

DESCRIPTION

Constructs the Accounting module of the SIP Server.

SYNTAX

```
RvStatus RvSipServerMgrConstructActModule(
    IN  RvSipServerMgrHandle              hServerMgr,
    IN  RvSipServerActLoggerCallbacks*    pActLoggerCb,
    IN  RvSipServerActCfg*                pCfg,
    OUT RvSipServerActMgrHandle*          phActMgr);
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

pActLoggerCb

A pointer to the Accounting logger callbacks structure.

pCfg

The address of the configuration structure.

phActMgr

The handle to the *AccountingMgr*.

RvSipServerMgrDestructActModule()

DESCRIPTION

Destruct the Accounting module of the SIP Server.

SYNTAX

```
RvStatus RvSipServerMgrDestructActModule(
    IN RvSipServerMgrHandle    hServerMgr);
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

RETURN VALUES

Returns RvStatus.

Get and Set Functions

The Get and Set functions are:

- RvSipServerMgrSetNewLogFilters()
- RvSipServerMgrDoesLogFilterExist()
- RvSipServerMgrGetLogHandle()
- RvSipServerMgrGetPolicyMgrHandle()
- RvSipServerMgrGetAuthMgrHandle()
- RvSipServerMgrGetRegServerMgrHandle()
- RvSipServerMgrGetServerDialogMgrHandle()
- RvSipServerMgrGetB2BMgrHandle()
- RvSipServerMgrGetPresMgrHandle()
- RvSipServerMgrGetListPool()
- RvSipServerMgrGetCallLegMgrHandle()
- RvSipServerMgrGetMsgMgrHandle()
- RvSipServerMgrGetTransportMgrHandle()
- RvSipServerMgrGetResources()
- RvSipServerMgrGetVersion()
- RvSipServerMgrGetStackVersion()
- RvSipServerMgrIsEnhancedDnsFeatureEnabled()
- RvSipServerMgrGetSeliHandle()
- RvSipServerMgrIsB2BFeatureEnabled()
- RvSipServerMgrIsServerEventsFeatureEnabled()
- RvSipServerMgrGetLocalAddress()
- RvSipServerMgrGetNumOfLocalAddress()
- RvSipServerMgrGetAppHandle()
- RvSipServerMgrIsTlsFeatureEnabled()
- RvSipServerMgrSetAppHandle()
- RvSipServerMgrGetServerTransportMgrHandle()
- RvSipServerMgrGetEventsMgrHandle()
- RvSipServerMgrGetWinfoMgrHandle()
- RvSipServerMgrGetXMLMgrHandle()
- RvSipServerMgrGetPublishMgrHandle()
- RvSipServerMgrGetPAMgrHandle()
- RvSipServerMgrGetLdapMgrHandle()

- RvSipServerMgrGetStartupConfig()
- RvSipServerMgrGetB2BAFMgrHandle()
- RvSipServerMgrGetB2BAFServiceLibMgrHandle()
- RvSipServerMgrGetActMgrHandle()
- RvSipServerMgrIsLDAPFeatureEnabled()
- RvSipServerMgrIsB2BAFFeatureEnabled()
- RvSipServerMgrIsSDPFeatureEnabled()
- RvSipServerMgrIsActFeatureEnabled()
- RvSipServerMgrGetResolverMgrHandle()
- RvSipServerMgrGetTransmitterMgrHandle()
- RvSipServerMgrGetRegClientMgrHandle()

RvSipServerMgrSetNewLogFilters()

DESCRIPTION

Sets the new SIP Server Log filters for the specified module during runtime.

SYNTAX

```
RvStatus RvSipServerMgrSetNewLogFilters(
    IN RvSipServerMgrHandle    hPSipServerMgr,
    IN RvSipServerModule       module,
    IN RvInt32                 filters);
```

PARAMETERS

hSipServerMgr

The handle to the SIP Server instance.

module

The module whose filters are going to be changed.

filters

The new set of filters. These filters will overwrite the filters that were previously set for this module.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrDoesLogFilterExist()

DESCRIPTION

Checks the existence of a filter for a given module. If the module is a SIP Stack module, RvSipStackIsLogFilterExist() is called.

SYNTAX

```
RvBool RvSipServerMgrDoesLogFilterExist(
    IN RvSipServerMgrHandle    hSipServerMgr,
    IN RvSipServerModule       module,
    IN RvUint8                 filters);
```

PARAMETERS

hSipServerMgr

The handle to the SIP Server instance.

module

The module whose filter is being checked.

filter

The filter that is being checked.

RETURN VALUES

If RV_TRUE, the filter exits for the given module. Otherwise, RV_FALSE.

RvSipServerMgrGetLogHandle()

DESCRIPTION

Returns the handle to the SIP Server Log instance.

SYNTAX

```
RvStatus RvSipServerMgrGetLogHandle(
    IN  RvSipServerMgrHandle    hSipServerMgr,
    OUT RV_LOG_Handle*          hLog);
```

PARAMETERS

hSipServerMgr

The handle to the SIP Server instance.

hLog

The handle to the SIP Server Log instance.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetPolicyMgrHandle()

DESCRIPTION

Returns the handle to the *PolicyMgr* object.

SYNTAX

```
RvStatus RvSipServerMgrGetPolicyMgrHandle(
    IN  RvSipServerMgrHandle       hSipServerMgr,
    OUT RvProxyPolicyMgrHandle*    hPolicyMgr);
```

PARAMETERS

hSipServerMgr

The handle to the SIP Server instance.

hPolicyMgr

The handle to the *PolicyMgr* object.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetAuthMgrHandle()

DESCRIPTION

Returns the handle to the *ServerAuthMgr* object.

SYNTAX

```
RvStatus RvSipServerMgrGetAuthMgrHandle(
    IN  RvSipServerMgrHandle         hSipServerMgr,
    OUT RvSipServerAuthMgrHandle*    hAuthMgr);
```

PARAMETERS

hSipServerMgr

The handler to the *SipServerMgr* object.

hAuthMgr

The handle to the *PolicyMgr* object.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetRegServerMgrHandle()

DESCRIPTION

Returns the handle to the *RegServerMgr* object.

SYNTAX

```
RvStatus RvSipServerMgrGetRegServerMgrHandle(
    IN  RvSipServerMgrHandle          hSipServerMgr,
    OUT RvProxyRegServerMgrHandle*    hRegServerMgr);
```

PARAMETERS

hSipServerMgr

The handle to the SIP Server instance.

hRegServerMgr

The handle to the *RegServerMgr* object.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetServerDialogMgrHandle()

DESCRIPTION

Returns the handle to the *ServerDialogMgr* object. This handle is required when the application is to register an event to the Server Dialog module.

SYNTAX

```
RvStatus RvSipServerMgrGetServerDialogMgrHandle(
    IN  RvSipServerMgrHandle           hSipServerMgr,
    OUT RvSipServerDialogMgrHandle*    phServerDialogMgr);
```

PARAMETERS

hSipServerMgr

The handler to the *SipServerMgr* object.

phServerDialogMgr

The handle to the *ServerDialogMgr* object.

RETURN VALUES

Returns RvStatus.

Note This function is relevant only if you have acquired the RADVISION SIP Server Back-to-Back (B2BUA) add-on module.

RvSipServerMgrGetB2BMgrHandle()

DESCRIPTION

Returns the handle to the *B2BMgr* object. This handle is required when the application is to register an event to the B2B module.

SYNTAX

```
RvStatus RvSipServerMgrGetB2BMgrHandle(
    IN  RvSipServerMgrHandle        hSipServerMgr,
    OUT RvSipServerB2BMgrHandle*    phB2BMgr);
```

PARAMETERS

hSipServerMgr

The handler to the *SipServerMgr* object.

phB2BMgr

The handle to the *B2BMgr* object.

RETURN VALUES

Returns RvStatus.

Note This function is relevant only if you have acquired the RADVISION SIP Server Back-to-Back (B2BUA) add-on module.

RvSipServerMgrGetPresMgrHandle()

DESCRIPTION

Returns the handle to the *PresMgr* object. This handle is required when the application is to register event to the Presence Server module.

SYNTAX

```
RvStatus RvSipServerMgrGetPresMgrHandle(
    IN  RvSipServerMgrHandle         hSipServerMgr,
    OUT RvSipServerPresMgrHandle*    phPresMgr);
```

PARAMETERS

hSipServerMgr

The handler to the *SipServerMgr* object.

phPresMgr

The handle to the *PresMgr* object.

RETURN VALUES

Returns RvStatus.

Note This function is relevant only if you have acquired the RADVISION SIP Events add-on module.

RvSipServerMgrGetListPool()

DESCRIPTION

Returns the handle to the SIP Server list pool.

SYNTAX

```
RvStatus RvSipServerMgrGetListPool(
    IN  RvSipServerMgrHandle       hSipServerMgr,
    OUT RvSipSrvListPoolHandle*    hSipSrvListPool);
```

PARAMETERS

hSipServerMgr

The handler to the *SipServerMgr* object.

hSipSrvListPool

The handle to the SIP Server list pool.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetCallLegMgrHandle()

DESCRIPTION

Returns the handle to the *CallLegMgr* object of the Stack. This handle is required when the application is to create a new Stack call-leg.

SYNTAX

```
RvStatus RvSipServerMgrGetCallLegMgrHandle(
    IN  RvSipServerMgrHandle      hSipServerMgr,
    OUT RvSipCallLegMgrHandle*    phCallLegMgr);
```

PARAMETERS

hSipServerMgr

The handler to the *SipServerMgr* object.

phCallLegMgr

The pointer handle to the *CallLegMgr* object.

RETURN VALUES

Returns RvStatus.

Note This function is relevant only if you have acquired the RADVISION SIP Server Back-to-Back (B2BUA) add-on module.

RvSipServerMgrGetMsgMgrHandle()

DESCRIPTION

Returns the handle to the *MsgMgr* object. This handle is required when the application wishes to construct a new message or header.

SYNTAX

```
RvStatus RvSipServerMgrGetMsgMgrHandle(
    IN  RvSipServerMgrHandle    hSipServerMgr,
    OUT RvSipMsgMgrHandle*      phMsgMgr);
```

PARAMETERS

hSipServerMgr

The handler to the *SipServerMgr* object.

phMsgMgr

The handle to the *MsgMgr* object.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetTransportMgrHandle()

DESCRIPTION

Returns the handle to the Transport Manager object (*TransportMgr*). This handle is required when the application is to work with the DNS List API.

SYNTAX

```
RvStatus RvSipServerMgrGetTransportMgrHandle(
    IN  RvSipServerMgrHandle        hSipServerMgr,
    OUT RvSipTransportMgrHandle*    phTransportMgr);
```

PARAMETERS

hSipServerMgr

The handler to the *SipServerMgr* object.

phTransportMgr

The handle to the *TransportMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetResources()

DESCRIPTION

Returns the resources object for a specified SIP Server module. If the module is a SIP Stack module, RvSipStackGetResources() is called.

SYNTAX

```
RvStatus RvSipServerMgrGetResources(
    IN  RvSipServerMgrHandle    hSipServer,
    IN  RvSipServerModule       module,
    OUT void*                   pResources);
```

PARAMETERS

hSipServer

The handle to the SIP Server instance.

module

The module whose resources are wanted.

pResources

The resources object of the module.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetVersion()

DESCRIPTION

Returns the SIP Server version number.

SYNTAX

```
RvChar* RvSipServerMgrGetVersion();
```

PARAMETERS

None.

RETURN VALUES

Returns the SIP Server version string.

RvSipServerMgrGetStackVersion()

DESCRIPTION

Returns the SIP Stack Toolkit version number.

SYNTAX

```
RvChar* RvSipServerMgrGetStackVersion();
```

PARAMETERS

None.

RETURN VALUES

Returns the SIP Stack Toolkit version string.

RvSipServerMgrIsEnhancedDnsFeatureEnabled()

DESCRIPTION

Indicates whether or not the SIP Server was compiled with or without the enhanced DNS feature

SYNTAX

```
RvStatus RvSipServerMgrIsEnhancedDnsFeatureEnabled(
   IN  RvSipServerMgrHandle  hSipServerMgr,
   OUT RvBool*               pbIsEnhancedDnsFeatureEnabled);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

pbIsEnhancedDnsFeatureEnabled

RV_TRUE if the SIP Server was compiled with DNS. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetSeliHandle()

DESCRIPTION

Gets the SELI (SELECT Interface) handle. You can use this handle with the SELI_CallOn() function to register an event on a file descriptor.

SYNTAX

```
RvStatus RvSipServerMgrGetSeliHandle(
    IN  RvSipServerMgrHandle    hSipServerMgr,
    OUT RV_SELI_Handle          *hSeli);
```

PARAMETERS

hSipServerMgr

The handle to the SIP Server instance.

hSeli

The handle to the SELI.

RETURN VALUES

Returns RvStatus.

REMARKS

You can use this function only if the SIP Server is compiled with the RV_DEPRECATED_CORE compilation flag, which means that the application is using the deprecated core implementation. However, it is not recommended to use the deprecated core. For low-level functionality, use the Middle Layer API.

RvSipServerMgrIsB2BFeatureEnabled()

DESCRIPTION

Indicates whether or not the SIP Server was compiled with the B2B add-on.

SYNTAX

```
RvStatus RvSipServerMgrIsB2BFeatureEnabled(
    IN  RvSipServerMgrHandle    hSipServerMgr,
    OUT RvBool*                 pbIsB2BFeatureEnabled);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

pbIsB2BFeatureEnabled

RV_TRUE if the SIP Server was compiled with the B2B add-on. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrIsServerEventsFeatureEnabled()

DESCRIPTION

Indicates whether or not the SIP Server was compiled with the Events Server Add-on Module.

SYNTAX

```
RvStatus RvSipServerMgrIsServerEventsFeatureEnabled(
  IN  RvSipServerMgrHandle  hSipServerMgr,
  OUT RvBool*               pbIsServerEventsFeatureEnabled);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

pbIsServerEventsFeatureEnabled

RV_TRUE if the SIP Server was compiled with the Events Server Add-on Module. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetLocalAddress()

DESCRIPTION

Copies a SIP Server local address and port according to the specified transport type and index. The amount of local addresses according to transport type can be retrieved using the RvSipServerMgrGetNumOfLocalAddress() function.

SYNTAX

```
RvStatus RvSipServerMgrGetLocalAddress(
    IN    RvSipServerMgrHandle    hSipServerMgr,
    IN    RvSipTransport          eTransoprtType,
    IN    RvInt32                 addressIndex,
    INOUT RvChar*                 pszRetAddress,
    OUT   RvUint16*               pPort);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

eTransoprtType

The requested transport type (UDP, TCP or TLS).

addressIndex

The requested address index.

pszRetAddress

The pointer to a buffer to which to copy the address. The buffer must be in the size of RVSIPSERVER_MAX_ADDRESS_LEN.

pPort

The returned address port.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetNumOfLocalAddress()

DESCRIPTION

Returns the amount of SIP Server local addresses according to the specified transport.

SYNTAX

```
RvInt32 RvSipServerMgrGetNumOfLocalAddress(
    IN RvSipServerMgrHandle    hSipServerMgr,
    IN RvSipTransport          eTransoprtType);
```

PARAMETERS

hSipServerMgr

The SIP Server handle.

eTransoprtType

The requested transport type (UDP, TCP or TLS).

RETURN VALUES

The amount of local addresses according to the specified transport type. The value of -1 is returned in case there is no local address of this type.

RvSipServerMgrGetAppHandle()

DESCRIPTION

Returns the handle to the application-associated object that is attached to the *SipServerMgr*.

SYNTAX

```
RvStatus RvSipServerMgrGetAppHandle(
    IN  RvSipServerMgrHandle     hSipServerMgr,
    OUT RvSipServerAppHandle*    phSipServerApp);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

phSipServerApp

The handle to the application object.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrIsTlsFeatureEnabled()

DESCRIPTION

Indicates whether or not the SIP Server was compiled with the TLS feature.

SYNTAX

```
RvStatus RvSipServerMgrIsTlsFeatureEnabled(
    IN  RvSipServerMgrHandle    hSipServerMgr,
    OUT RvBool*                 pbIsTlsFeatureEnabled);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

pbIsTlsFeatureEnabled

A flag that indicates whether or not the SIP Server was compiled with TLS.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrSetAppHandle()

DESCRIPTION

Attaches an application-associated object to the *SipServerMgr*.

SYNTAX

```
RvStatus RvSipServerMgrSetAppHandle(
    IN RvSipServerMgrHandle    hSipServerMgr,
    IN RvSipServerAppHandle    hSipServerApp);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

hSipServerApp

The handle to the application object.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetServerTransportMgrHandle()

DESCRIPTION

Returns the handle to the Server Transport Manager object (*ServerTransportMgr*).

SYNTAX

```
RvStatus RvSipServerMgrGetServerTransportMgrHandle(
  IN  RvSipServerMgrHandle            hSipServerMgr,
  OUT RvSipServerTransportMgrHandle*  phServerTransportMgr);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

phServerTransportMgr

The handle to the *ServerTransportMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetEventsMgrHandle()

DESCRIPTION

Returns the handle to the Events Manager object (*EventsMgr*). This handle is required when the application wants to register events to the Events module.

SYNTAX

```
RvStatus RvSipServerMgrGetEventsMgrHandle(
    IN  RvSipServerMgrHandle           hSipServerMgr,
    OUT RvSipServerEventsMgrHandle*    phEventsMgr);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

phEventsMgr

The handle to the *EventsMgr*.

RETURN VALUES

Returns RvStatus.

REMARKS

This function is relevant only if you have acquired the RADVISION SIP Server Events add-on module.

RvSipServerMgrGetWinfoMgrHandle()

DESCRIPTION

Returns the handle to the Winfo Manager object (*WinfoMgr*). This handle is required when the application wants to register events to the Winfo module.

SYNTAX

```
RvStatus RvSipServerMgrGetWinfoMgrHandle(
    IN  RvSipServerMgrHandle          hSipServerMgr,
    OUT RvSipServerWinfoMgrHandle*    phWinfoMgr);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

phWinfoMgr

The handle to the *WinfoMgr*.

RETURN VALUES

Returns RvStatus.

REMARKS

This function is relevant only if you have acquired the RADVISION SIP Server Events add-on module.

RvSipServerMgrGetXMLMgrHandle()

DESCRIPTION

Returns the handle to the XML Manager object (*XMLMgr*). This handle is required when the application wants to encode a new XML document.

SYNTAX

```
RvStatus RvSipServerMgrGetXMLMgrHandle(
    IN  RvSipServerMgrHandle    hSipServerMgr,
    OUT RvXMLMgrHandle*         phXMLMgr);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

phXMLMgr

The handle to the *XMLMgr*.

RETURN VALUES

Returns RvStatus.

REMARKS

This function is relevant only if you have acquired the RADVISION SIP Server Events add-on module.

RvSipServerMgrGetPublishMgrHandle()

DESCRIPTION

Returns the handle to the Publish Manager object (*PublishMgr*). This handle is required when the application wants to register events to the Publish module.

SYNTAX

```
RvStatus RvSipServerMgrGetPublishMgrHandle(
    IN  RvSipServerMgrHandle            hSipServerMgr,
    OUT RvSipServerPublishMgrHandle*    phPublishMgr);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

phPublishMgr

The handle to the *PublishMgr*.

RETURN VALUES

Returns RvStatus.

REMARKS

This function is relevant only if you have acquired the Events Server Add-on Module.

RvSipServerMgrGetPAMgrHandle()

DESCRIPTION

Returns the handle to the Presence Agent Manager object (*PAMgr*). This handle is required when the application wants to register events to the Presence Agent module.

SYNTAX

```
RvStatus RvSipServerMgrGetPAMgrHandle(
    IN  RvSipServerMgrHandle       hSipServerMgr,
    OUT RvSipServerPAMgrHandle*    phPAMgr);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

phPAMgr

The handle to the *PAMgr*.

RETURN VALUES

Returns RvStatus.

REMARKS

This function is relevant only if you have acquired the Events Server Add-on Module.

RvSipServerMgrGetLdapMgrHandle()

DESCRIPTION

Returns the handle to the LDAP Manager object (*LDAPMgr*). This handle is required when the application wants to register events to the LDAP module.

SYNTAX

```
RvStatus RvSipServerMgrGetLdapMgrHandle(
    IN  RvSipServerMgrHandle         hSipServerMgr,
    OUT RvSipServerLdapMgrHandle*    phLdapMgr);
```

PARAMETERS

hSipServerMgr

The *SipServerMgr* handle.

phLdapMgr

The handle to the *LDAPMgr*.

RETURN VALUES

Returns RvStatus.

REMARKS

This function is relevant only if you have acquired the Events Server Add-on Module.

RvSipServerMgrGetStartupConfig()

DESCRIPTION

Returns the configuration parameters specified in the RvSipServerCfg structure, or in the configuration file, with the SIP Server default values that were used as the startup configuration of the SIP Server.

SYNTAX

```
RvStatus RvSipServerMgrGetStartupConfig(
    IN    RvSipServerMgrHandle    hSipServerMgr,
    INOUT RvInt32                 sizeOfCfg,
    OUT   RvSipServerCfg*         pServerCfg);
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

sizeOfCfg

The size of the configuration structure.

pServerCfg

The startup configuration structure containing the SIP Server default values.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetB2BAFMgrHandle()

DESCRIPTION

Returns the handle to the B2BAF Manager object (*B2BAFMgr*). This handle is required when the application wants to register an event to the B2BAF module.

SYNTAX

```
RvStatus RvSipServerMgrGetB2BAFMgrHandle(
    IN  RvSipServerMgrHandle          hSipServerMgr,
    OUT RvSipServerB2BAFMgrHandle*    phB2BAFMgr);
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

phB2BAFMgr

The returned handle to the *B2BAFMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetB2BAFServiceLibMgrHandle()

DESCRIPTION

Returns the handle to the B2BAF Service Library Manager object (*ServiceLibMgr*).

SYNTAX

```
RvStatus RvSipServerMgrGetB2BAFServiceLibMgrHandle(
    IN  RvSipServerMgrHandle    hSipServerMgr,
    OUT RvSipServerB2BAFServiceLibMgrHandle*
                                phB2BAFServiceLibMgr);
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

phB2BAFServiceLibMgr

The returned handle to the *ServiceLibMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetActMgrHandle()

DESCRIPTION

Returns the handle to the Accounting Manager object (*AccountingMgr*).

SYNTAX

```
RvStatus RvSipServerMgrGetActMgrHandle(
    IN  RvSipServerMgrHandle        hSipServerMgr,
    OUT RvSipServerActMgrHandle*    phActMgr);
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

phActMgr

The handle to the *AccountingMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrIsLDAPFeatureEnabled()

DESCRIPTION

Indicates whether the SIP Server was compiled with or without the LDAP Add-on Module.

SYNTAX

```
RvStatus RvSipServerMgrIsLDAPFeatureEnabled(
    IN  RvSipServerMgrHandle    hSipServerMgr,
    OUT RvBool*                 pbIsLDAPFeatureEnabled);
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

pbIsLDAPFeatureEnabled

RV_TRUE if the SIP Server was compiled with LDAP, otherwise RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrIsB2BAFFeatureEnabled()

DESCRIPTION

Indicates whether the SIP Server was compiled with or without the B2BAF Add-on Module.

SYNTAX

```
RvStatus RvSipServerMgrIsB2BAFFeatureEnabled(
    IN  RvSipServerMgrHandle    hSipServerMgr,
    OUT RvBool*                 pbIsB2BAFFeatureEnabled);
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

pbIsB2BAFFeatureEnabled

RV_TRUE of the SIP Server was compiled with B2BAF, otherwise RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrIsSDPFeatureEnabled()

DESCRIPTION

Indicates whether the SIP Server was compiled with or without the SDP Add-on Module.

SYNTAX

```
RvStatus RvSipServerMgrIsSDPFeatureEnabled(
    IN  RvSipServerMgrHandle    hSipServerMgr,
    OUT RvBool*                 pbIsSDPFeatureEnabled);
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

pbIsSDPFeatureEnabled

RV_TRUE if the SIP Server was compiled with SDP, otherwise RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrIsActFeatureEnabled()

DESCRIPTION

Indicates whether the SIP Server was compiled with or without the Accounting Add-on Module.

SYNTAX

```
RvStatus RvSipServerMgrIsActFeatureEnabled(
    IN  RvSipServerMgrHandle    hSipServerMgr,
    OUT RvBool*                 pbIsActFeatureEnabled);
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

pbIsActFeatureEnabled

RV_TRUE if the SIP Server was compiled with the Accounting Module, otherwise RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetResolverMgrHandle()

DESCRIPTION

Returns the handle to the Resolver Manager object (*ResolverMgr*). This handle is required when the application wants to construct a new Resolver and perform a DNS query.

SYNTAX

```
RvStatus RvSipServerMgrGetResolverMgrHandle(
    IN  RvSipServerMgrHandle         hSipServerMgr,
    OUT RvSipResolverMgrHandle*    phResolverMgr);
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

phResolverMgr

The handle to the *ResolverMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetTransmitterMgrHandle()

DESCRIPTION

Returns the handle to the SIP Toolkit Transmitter Manager object (*TransmitterMgr*). This handle is required when the application wishes to create a *transmitter* object.

SYNTAX

```
RvStatus RvSipServerMgrGetTransmitterMgrHandle(
    IN  RvSipServerMgrHandle          hSipServerMgr,
    OUT RvSipTransmitterMgrHandle*    phTransmitterMgr)
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

phTransmitterMgr

The returned handle to the *TransmitterMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerMgrGetRegClientMgrHandle()

DESCRIPTION

Returns the handle to the RegClient Manager object (*RegClientMgr*). This handle is required when the application wants to get a *register-client* resource.

SYNTAX

```
RvStatus RvSipServerMgrGetRegClientMgrHandle(
    IN  RvSipServerMgrHandle        hSipServerMgr,
    OUT RvSipRegClientMgrHandle*    phRegClientMgr)
```

PARAMETERS

hSipServerMgr

The handle to the *SIPServerMgr*.

phRegClientMgr

The handle to the *RegClientMgr*.

RETURN VALUES

Returns RvStatus.

Get and Set Functions
RvSipServerMgrGetRegClientMgrHandle()

PROXY CORE MODULE

The Proxy Core module enables you to control the Proxy Core Objects (*ProxyCoreObj*), Proxy Core Transaction objects (*ProxyCoreTransc*), and Proxy Core Transmitter objects (*ProxyTrx*).

Using the Proxy Core Object API functions, you can choose to redirect, reject or proxy an incoming request, in sequential or parallel forking. You will receive notification about incoming responses and you can decide how to handle these responses. The Proxy Core Object API functions and callbacks enable you to work in a transaction stateful or stateless mode.

This part includes the following section:

2

PROXY CORE FUNCTIONS

WHAT'S IN THIS SECTION

This section contains the Proxy Core functions, which are grouped as follows:

- Proxy Core Policy Manager Functions
- Proxy Core Object Functions
- Proxy Core Transaction Functions
- Proxy Core Transmitter Functions

Proxy Core Policy Manager Functions

The Proxy Core Policy Manager (*ProxyPolicyMgr*) is responsible for Proxy Core Objects (*ProxyCoreObj*), Proxy Core Transaction objects (*ProxyCoreTransc*), Proxy Core Transmitter objects (*ProxyTrx*), and Proxy Server Authentication objects (*ProxyServerAuthObj*). The Proxy Core Policy Manager API functions enable you to set the event handler for the *ProxyCoreObj* objects. These functions are included in the *RvProxyCorePolicyMgr.h* header file.

The Proxy Core Policy Manager API includes:

- Control Functions
- Get and Set Functions

Control Functions

The Control functions are:

- RvProxyCorePolicyMgrAttachAppObj()
- RvProxyCorePolicyMgrCreateSFCoreObj()
- RvProxyCorePolicyMgrCreateSLCoreObj()

RvProxyCorePolicyMgrAttachAppObj()

DESCRIPTION

Attaches the application-associated object to the *ProxyPolicyMgr*.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrAttachAppObj(
    IN RvProxyPolicyMgrHandle       hMgr,
    IN RvProxyPolicyMgrAppHandle    hApp);
```

PARAMETERS

hMgr

The *ProxyPolicyMgr*.

hAppbd

The handle of application-associated object to attach to the *ProxyPolicyMgr*.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrCreateSFCoreObj()

DESCRIPTION

Creates a new, stateful *ProxyCoreObj* with a client *ProxyCoreTransc* and associates the application object to the new object. The application handles that are given as input parameters will be supplied at all callbacks that are relevant to this object.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrCreateSFCoreObj(
    IN    RvProxyPolicyMgrHandle     hPolicyMgr,
    IN    RvProxyCoreObjAppHandle    hAppCoreObj,
    IN    RvProxyTranscAppHandle     hAppTransc,
    INOUT RvProxyCoreObjHandle*      phCoreObj,
    INOUT RvProxyTranscHandle*       phTransc);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

hAppCoreObj

The application object to associate to the *ProxyCoreObj*.

hAppTransc

The application object to associate to the *ProxyCoreTransc*.

phCoreObj

The new *ProxyCoreObj* that was created.

phTransc

The new *ProxyCoreTransc* that was created.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrCreateSLCoreObj()

DESCRIPTION

Creates a new stateless *ProxyCoreObj* with a *ProxyTrx* and associates the application object to the new object. The application handles that are given as input parameters will be supplied at all callbacks that are relevant to this object.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrCreateCoreObj(
    IN    RvProxyPolicyMgrHandle     hPolicyMgr,
    IN    RvProxyCoreObjAppHandle    hAppCoreObj,
    IN    RvProxyTrxAppHandle        hAppTrx,
    INOUT RvProxyCoreObjHandle*      phCoreObj,
    INOUT RvProxyTrxHandle*          phProxyTrx);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

hAppCoreObj

The application object to associate to the *ProxyCoreObj*.

hAppTrx

The application object to associate to the *ProxyTrx*.

phCoreObj

The new *ProxyCoreObj* that was created.

phProxyTrx

The new *ProxyTrx* that was created.

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

The Get and Set functions are:

- RvProxyCorePolicyMgrSetStatefulEvHandler()
- RvProxyCorePolicyMgrSetCoreObjEvHandler()
- RvProxyCorePolicyMgrSetPolicyMgrEvHandler()
- RvProxyCorePolicyMgrGetAppObj()
- RvProxyCorePolicyMgrGetResourceStatus()
- RvProxyCorePolicyMgrGetServerInstance()
- RvProxyCorePolicyMgrSetStatelessEvHandler()
- RvProxyCorePolicyMgrGetDnsDomains()
- RvProxyCorePolicyMgrSetDnsDomains()
- RvProxyCorePolicyMgrGetDnsServers()
- RvProxyCorePolicyMgrSetDnsServers()

RvProxyCorePolicyMgrSetStatefulEvHandler()

DESCRIPTION

Sets the module callback function for a transaction stateful SIP Server.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrSetStatefulEvHandler(
    IN RvProxyCoreStateFulEvHandler*    pEvHandler,
    IN RvProxyPolicyMgrHandle           hMgr,
    IN RvUint32                         size);
```

PARAMETERS

pEvHandler

A pointer to transaction stateful SIP Server event handler.

hMgr

The handle of the *ProxyPolicyMgr* to which to set the event handler.

size

The size of the RvProxyCoreStateFulEvHandler structure.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrSetCoreObjEvHandler()

DESCRIPTION

Sets the module callback function for events that are relevant to both Transaction stateful and stateless proxy.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrSetCoreObjEvHandler(
    IN RvProxyCoreObjEvHandler    *pEvHandler,
    IN RvProxyPolicyMgrHandle     hMgr,
    IN RvUint32                   size);
```

PARAMETERS

pEvHandler

A pointer to *CoreObj* event handler.

hMgr

The handle of the *ProxyPolicyMgr* to which to set the event handler.

size

The size of the RvProxyCoreObjEvHandler structure.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrSetPolicyMgrEvHandler()

DESCRIPTION

Sets the module callback functions for the *ProxyPolicyMgr* events.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrSetPolicyMgrEvHandler(
    IN RvPolicyMgrEvHandler*     pEvHandler,
    IN RvProxyPolicyMgrHandle    hMgr,
    IN RvUint32                  size);
```

PARAMETERS

pEvHandler

A pointer to *ProxyPolicyMgr* events handler.

hMgr

The handle of the *ProxyPolicyMgr* to which to set the event handler.

size

The size of the RvPolicyMgrEvHandler structure.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrGetAppObj()

DESCRIPTION

Gets the handle of the application *ProxyPolicyMgr*.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrGetAppObj(
    IN  RvProxyPolicyMgrHandle        hPolicyMgr,
    OUT RvProxyPolicyMgrAppHandle*    phAppPolicyMgr);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

phAppPolicyMgr

The handle of the application *ProxyPolicyMgr*.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrGetResourceStatus()

DESCRIPTION

Returns the resource status of the Policy Manager module.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrGetResourceStatus(
    IN  RvProxyPolicyMgrHandle    hPolicyMgr,
    OUT RvProxyPolicyResources    *pResourcesStatus);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

pResourcesStatus

The resource structure.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrGetServerInstance()

DESCRIPTION

Gets the handle of the *ProxyPolicyMgr*.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrGetServerInstance(
    IN  RvProxyPolicyMgrHandle    hPolicyMgr,
    OUT void**                    phSipServerMgr);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

phSipServerMgr

The handle of the *SIPServerMgr*.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrSetStatelessEvHandler()

DESCRIPTION

Sets the module callback function for a stateless SIP Server.

SYNTAX

```
Rvstatus RvProxyCorePolicyMgrSetStatelessEvHandler(
    IN RvProxyTrxSLEvHandler*    pEvHandler,
    IN RvProxyPolicyMgrHandle    hMgr,
    IN RvUint32                  size);
```

PARAMETERS

pEvHandler

A pointer to *ProxyTrx* stateless event handler.

hMgr

The handle of the *ProxyPolicyMgr* to which to set the event handler.

size

The size of the RvProxyTrxStatelessEvHandler structure.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrGetDnsDomains()

DESCRIPTION

Gets the list of DNS domains from the SIP Server. This function is useful to determine the list of DNS domains the with which the SIP Server started.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrGetDnsDomains(
    IN    RvProxyPolicyMgrHandle    hPolicyMgr,
    INOUT RvInt32*                  pNumOfDomains,
    OUT   RvChar**                  pDomainList);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

pNumOfDomains

The size of the *pDomainList* array. The SIP Server will return the number of domains that were actually set.

pDomainsList

A list of DNS domains (an array of char pointers). This array will be filled with pointers to DNS domains. The list of DNS domains is part of the SIP Server memory. If the application wishes to manipulate this list, it must copy the strings to a different memory space. The size of this array must be no smaller than *pnumOfDomains*.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrSetDnsDomains()

DESCRIPTION

Sets a new list of DNS domains to the SIP Server. The SIP Server provides a Domain Suffix Search Order capability. The Domain Suffix Search Order specifies the DNS domain suffixes to be appended to the host names during name resolution. When attempting to resolve a fully qualified domain name (FQDN) from a host that includes a name only, the system will first append the local domain name. If this is not successful, the system will use the Domain Suffix list to create additional FQDNs in the order listed and query DNS servers for each. When the SIP Server starts, the DNS domain list is set by the configuration of the computer. The application can use this function to provide a new set of DNS domains to the SIP Server.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrSetDnsDomains(
    IN RvProxyPolicyMgrHandle    hPolicyMgr,
    IN RvChar**                  pDomainList,
    IN RvInt32                   numOfDomains);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

pDomainsList

A list of DNS domains (an array of NULL terminated strings) to set to the SIP Server.

numOfDomains

The number of domains in the list.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrGetDnsServers()

DESCRIPTION

Gets the list of DNS servers from the SIP Server. This function is useful to determine the list of DNS servers with which the SIP Server started.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrGetDnsServers(
    IN    RvProxyPolicyMgrHandle    hPolicyMgr,
    INOUT RvInt32*                  pNumOfDnsServers,
    OUT   RvSipTransportAddr*       pDnsServers);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

pNumOfDomains

The size of *pDnsServers*. This value will be updated with the actual size of the DNS servers list.

pDnsServers

An empty list of DNS servers that will be filled with the DNS servers that the SIP Server is currently using.

RETURN VALUES

Returns RvStatus.

RvProxyCorePolicyMgrSetDnsServers()

DESCRIPTION

Sets a new list of DNS servers to the SIP Server.When the SIP Server starts, the DNS list is set by the configuration of the computer. The application can use this function to provide a new set of DNS servers to the SIP Server.

SYNTAX

```
RvStatus RvProxyCorePolicyMgrSetDnsServers(
    IN RvProxyPolicyMgrHandle    hPolicyMgr,
    IN RvSipTransportAddr*       pDnsServers,
    IN RvInt32                   numOfDnsServers);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

pDnsServers

A list of DNS servers to set to the SIP Server.

numOfDomains

The number of DNS servers in the list.

RETURN VALUES

Returns RvStatus.

Proxy Core Object Functions

The Proxy Core Object API functions enable you to handle received requests according to the application policy, to handle received responses and access the *ProxyCoreObj* parameters. These functions are included in the *RvProxyCoreObj.h* header file.

The Proxy Core Object API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvProxyCoreObjTerminate()
- RvProxyCoreObjDisregardInvalidRequest()
- RvProxyCoreObjAcceptRequest()
- RvProxyCoreObjRejectRequest()
- RvProxyCoreObjectSendProvisionalResponse()
- RvProxyCoreObjResolveAddr()
- RvProxyCoreObjRedirectRequest()
- RvProxyCoreObjProxyRequest()
- RvProxyCoreObjCancel()
- RvProxyCoreObjDetermineRequestTarget()
- RvProxyCoreObjGetReceivedFromAddress()
- RvProxyCoreObjRouteInfoPreprocess()

RvProxyCoreObjTerminate()

DESCRIPTION

Terminates a *ProxyCoreObj* and frees all resources allocated by it, such as messages, *ProxyCoreTransc* objects, *ProxyTrx* objects and the address list. The *ProxyCoreObj* will assume the TERMINATE state.

SYNTAX

```
RvStatus RvProxyCoreObjTerminate(
    IN RvProxyCoreObjHandle    hCoreObj);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to terminate.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjDisregardInvalidRequest()

DESCRIPTION

Disregards the request validation failure. On receipt of a request, the request validation process is performed according to the *ProxyCoreObj* attribute. If a request validation has failed, the *ProxyCoreObj* assumes the INVALID_REQUEST or AUTHENTICATION_FAILED state. If the application wishes to ignore this error, it can call this function and the *ProxyCoreObj* will continue with the request process. This function can be called on any reason except LOOP_DETECTED.

SYNTAX

```
RvStatus RvProxyCoreObjDisregardInvalidRequest(
    IN RvProxyCoreObjHandle    hCoreObj);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* that disregards the validation failure.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjAcceptRequest()

DESCRIPTION

Creates and sends a 2xx response message.

SYNTAX

```
RvStatus RvProxyCoreObjAcceptRequest(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvUint16                respCode,
    IN const RvChar*           szReasonPhrase)
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* that will accept the request.

respCode

The response code which must be between 200 and 299.

szReasonPhrase

A NULL terminated string that contains the accept reason. If *szReasonPhrase* is NULL, the default accept reason is used.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjRejectRequest()

DESCRIPTION

Creates and sends a Reject response message. If the application wishes to reject an invalid request (a *ProxyCoreObj* that reached the INVALID_REQUEST state), it can use a value of 0 in the *respCode* parameter and a reject message with a response code that matches the reason for the request validation failure will be sent. If 0 is used for the *respCode* for a state other than INVALID, a default 400 response will be sent. For example, a *ProxyCoreObj* reached the INVALID_REQUEST state with a reason of Illegal Max Forward. If the application rejects the request and the *respCode* value is 0, a 483 response message will be generated and sent.

SYNTAX

```
RvStatus RvProxyCoreObjRejectRequest(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvUint16                respCode,
    IN const RvChar            *szReasonPhrase);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* that sends the Reject message.

respCode

The response code of the Reject message. Note the following:

- If this parameter is 0 and the state is INVALID, a response code that matches the reason for the request validation failure will be sent.
- If this parameter is not 0, it must be between 400-699, and this value will be used as a response code.
- If the state is not INVALID, a default response code (400) will be sent.

szReasonPhrase

A NULL terminated string containing the reason for the rejection. If this parameter is NULL, the default reject reason is used.

Return Values

Returns RvStatus.

RvProxyCoreObjectSendProvisionalResponse()

DESCRIPTION

Sends a provisional response. The response code must be 100. The application can use this function only in the REQUEST_RECEIVED state.

SYNTAX

```
RvStatus RvProxyCoreObjectSendProvisionalResponse(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvUint16                respCode,
    IN const RvChar            *szReasonPhrase);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* that sends the provisional response.

respCode

The provisional response code which must be 100.

szReasonPhrase

A NULL terminated string containing the provisional response reason. If this parameter is NULL, the default reason is used.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjResolveAddr()

DESCRIPTION

Resolves the destination address of the receive request. The application can use this function only in the REQUEST_VALID state.

SYNTAX

```
RvStatus RvProxyCoreObjResolveAddr(
    IN RvProxyCoreObjHandle    hCoreObj);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* whose request destination address should be resolved.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjRedirectRequest()

DESCRIPTION

Sends a 3xx redirection response. The application uses this function only in the following states:

- RVPROXY_CORE_OBJ_STATE_REQUEST_RECEIVED
- RVPROXY_CORE_OBJ_STATE_REQUEST_VALID
- RVPROXY_CORE_OBJ_STATE_ADDRESS_RESOLVED
- RVPROXY_CORE_OBJ_STATE_DESTINATION_ADDRESS_NOT_FOUND
- RVPROXY_CORE_OBJ_STATE_INVALID_REQUEST
- RVPROXY_CORE_OBJ_STATE_AUTHENTICATION_FAILED

The application needs to supply a contact list to add to this redirection response. The application can use this function only with 300 to 399 response codes, and in the transaction stateful mode only.

SYNTAX

```
RvStatus RvProxyCoreObjRedirectRequest(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvUint16                respCode,
    IN const RvChar*           strReasonPhrase,
    IN RvProxyListHandle       hContactsList);
```

PARAMETERS

hCoreObj

The redirected *ProxyCoreObj*.

respCode

The 3xx response code.

strReasonPhrase

A NULL terminated string to add as a reason phrase to the response. If NULL, a response phrase default reason phrase is not added.

hContactsList

A list of contacts to insert into the redirection response.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjProxyRequest()

DESCRIPTION

Forwards incoming request with or without forking the request. If the application wishes to modify the destAddressList, it should change the destAddressList of the *ProxyCoreObj* before calling this function. If the application does not modify the destAddressList, the request is proxied to the addresses resolved by the proxy at the routing decision process.

Concerning the forwarding of a stateless request, if the request was forwarded successfully, the *ProxyCoreObj* is terminated after forwarding the request. If a network error occurred while forwarding the request, the *ProxyCoreObj* initiates a 500 final response to the UAC and the *ProxyCoreObj* becomes stateful. In this case, the function returns RV_NETWORK_PROBLEM.

This function is valid in the following state:

- RVPROXY_CORE_OBJ_STATE_ADDRESS_RESOLVED

SYNTAX

```
RvStatus RvProxyCoreObjProxyRequest(
    IN RvProxyCoreObjHandle      hCoreObj,
    IN RvProxyCoreForkingType    forkingType);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to which the request should be forwarded.

forkingType

Specifies whether the request should be forward sequentially—sending requests to the addresses one by one, or in parallel—sending requests all at once to all addresses in the *hDestAddrList* of the *ProxyCoreObj*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjCancel()

DESCRIPTION

Sends a CANCEL on the specified *ProxyCoreObj*. If the request has been forked, a CANCEL will be called on each of the pending *ProxyCoreTransc* objects. The application can use this function only in the SEQ_PROXYING_REQUEST and PARALLEL_PROXYING_REQUEST states.

Note The *ProxyCoreObj* state will be changed to REQUEST_CANCELLED only if at least one transaction was actually cancelled.

SYNTAX

```
RvStatus RvProxyCoreObjCancel(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvBool                  bCheckTranscProceeding);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to cancel.

bCheckTranscProceeding

Specifies whether to check if the transaction is proceeding (received 1xx) before cancelling.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjDetermineRequestTarget()

DESCRIPTION

Performs the “Determine request target” procedure of RFC 3261, section 16.4. This function determines the final destination of the request.

SYNTAX

```
RvProxyCoreObjDetermineRequestTarget(
    IN  RvProxyCoreObjHandle    hCoreObj);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to set this attribute.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjGetReceivedFromAddress()

DESCRIPTION

Gets the address from which the last message was received.

SYNTAX

```
RvProxyCoreObjGetReceivedFromAddress(
    IN  RvProxyCoreObjHandle    hCoreObj,
    OUT RvSipTransportAddr      *pAddr)
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to set this attribute.

pAddr

The basic details about the received From address.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjRouteInfoPreprocess()

DESCRIPTION

Performs the “Route information processing” procedure of RFC 3261, section 16.4, and message conversion strict and loose routing.

SYNTAX

```
RvProxyCoreObjRouteInfoPreprocess(
    IN  RvProxyCoreObjHandle    hCoreObj);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to set this attribute.

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

The Get and Set functions are:

- RvProxyCoreObjGetReceivedRequest()
- RvProxyCoreObjSetDestAddress()
- RvProxyCoreObjGetResolvedAddressList()
- RvProxyCoreObjGetCancelPair()
- RvProxyCoreObjSetAutoTry()
- RvProxyCoreObjSetAuthRequired()
- RvProxyCoreObjSetRecurseOn3xx()
- RvProxyCoreObjSetLoopDetectionRequired()
- RvProxyCoreObjGetRequestInvalidReason()
- RvProxyCoreObjSetRecordRouteMode()
- RvProxyCoreObjSetViaHeader()
- RvProxyCoreObjGetState()
- RvProxyCoreObjGetMode()
- RvProxyCoreObjGetPolicyMgr()
- RvProxyCoreObjGetReceivedLocalAddress()
- RvProxyCoreObjAddRoutesToRequest()
- RvProxyCoreObjSetSTPreferenceParams()
- RvProxyCoreObjAddAuthHeaderToResponse()
- RvProxyCoreObjServerTranscSetTimers()

RvProxyCoreObjGetReceivedRequest()

DESCRIPTION

Returns the received request message. This function succeeds even if no request message has been found, therefore *phMsg* must be inspected before further usage. The last time that the received request message is valid is when the *ProxyCoreObj* reports the RVPROXY_CORE_OBJ_STATE_REQUEST_VALID, RVPROXY_CORE_OBJ_STATE_INVALID_REQUEST or RVPROXY_CORE_OBJ_STATE_AUTHENTICATION_FAILED states using the RvProxyCoreObjStateChangeEv() callback. If the application wishes to use this massage later on, the application must copy it.

This function is valid in the following states of the *ProxyCoreObj*:

- RVPROXY_CORE_OBJ_STATE_IDLE
- RVPROXY_CORE_OBJ_STATE_REQUEST_RECEIVED
- RVPROXY_CORE_OBJ_STATE_AUTHENTICATING
- RVPROXY_CORE_OBJ_STATE_AUTHENTICATION_FAILED
- RVPROXY_CORE_OBJ_STATE_INVALID_REQUEST
- RVPROXY_CORE_OBJ_STATE_REQUEST_VALID

SYNTAX

```
RvStatus RvProxyCoreObjGetReceivedRequest(
    IN  RvProxyCoreObjHandle    hCoreObj,
    OUT RvSipMsgHandle          *phMsg);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* whose received request is returned.

phMsg

The returned received request message.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjSetDestAddress()

DESCRIPTION

Sets a destination address to a *ProxyCoreObj*. If the *ProxyCoreObj* assumes the DESTINATION_ADDRESS_NOT_FOUND state, and the application wishes to forward the request to one or more destinations, it can set the *ProxyCoreObj* destination address using this function.

Calling this function in the following states is legal:

- ADDRESS_RESOLVED
- DESTINATION_ADDRESS_NOT_FOUND

SYNTAX

```
RvStatus RvProxyCoreObjSetDestAddress(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvProxyListHandle       hDestAddrList);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to set its destination address.

hDestAddrList

The handle to address list to set.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjGetResolvedAddressList()

DESCRIPTION

Returns the *ProxyCoreObj* resolved address list. This function returns a pointer to internal member. The application *must not* free this list—it *may* remove or add address elements to it. If this function is called before the *ProxyCoreObj* reached the ADDRESS_RESOLVED state, the returned destination address list will be NULL.

SYNTAX

```
RvStatus RvProxyCoreObjGetResolvedAddressList(
    IN  RvProxyCoreObjHandle    hCoreObj,
    OUT RvProxyListHandle       *phDestAddrList);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to return its resolved address list.

phDestAddrList

The pointer to the resolved address list address.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjGetCancelPair()

DESCRIPTION

Returns the *ProxyCoreObj* cancel pair. If no cancel pair was found, *hCancelPair* is NULL.

SYNTAX

```
RvStatus RvProxyCoreObjGetCancelPair(
    IN  RvProxyCoreObjHandle    hCoreObj,
    OUT RvProxyCoreObjHandle    *phCancelPair);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to return its cancel pair.

phCancelPair

The returned cancel pair.

RETURN VALUES

Returns RvStatus.

REMARKS

A “cancel pair” consists of two elements:

- The *ProxyCoreObj* of the CANCEL request.
- The *ProxyCoreObj* to be cancelled.

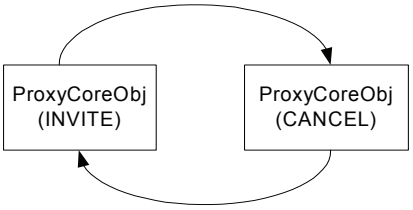

Figure 2-1 *Cancel Pair*

RvProxyCoreObjSetAutoTry()

DESCRIPTION

Indicates whether or not the specify *ProxyCoreObj* should automatically send a 100 provisional response on receipt of an INVITE request.

SYNTAX

```
RvStatus RvProxyCoreObjSetAutoTry(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvBool                  bIsAutoTry);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to which to set this attribute.

bIsAutoTry

RV_TRUE if the object will send a 100 response on receipt of an INVITE request. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjSetAuthRequired()

DESCRIPTION

Indicates whether or not the specified *ProxyCoreObj* should use the authentication mechanism on any received request (except CANCEL and ACK).

SYNTAX

```
RvStatus RvProxyCoreObjSetAuthRequired(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvBool                  bIsAuthentication);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to set the authenticate request attribute.

bIsAuthentication

RV_TRUE if the object should use authentication. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjSetRecurseOn3xx()

DESCRIPTION

Indicates whether or not the specified *ProxyCoreObj* should recurse a 3xx response.

SYNTAX

```
RvStatus RvProxyCoreObjSetRecurseOn3xx(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvBool                  bIsRecurse3xx);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to set the recurse on 3xx attribute.

bIsRecourse3xx

RV_TRUE if the object should recurse 3xx responses. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjSetLoopDetectionRequired()

DESCRIPTION

Indicates whether or not the specified *ProxyCoreObj* should perform a loop detection on the received request.

SYNTAX

```
RvStatus RvProxyCoreObjSetLoopDetectionRequired(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvBool                  bIsLoopDetectionRequire);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to set the attribute.

bIsLoopDetectionRequire

RV_TRUE if the object will perform the loop detection. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjGetRequestInvalidReason()

DESCRIPTION

Gets the reason for the request validation failure and response code which matches this validation failure. If the request is valid, the function returns UNDEFINED on *peReason* and 0 on *pRespCode*.

SYNTAX

```
RvStatus RvProxyCoreObjGetRequestInvalidReason(
    IN  RvProxyCoreObjHandle    hCoreObj,
    OUT RvProxyCoreObjReason    *peReason,
    OUT RvUint16                *pRespCode);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to get the reason for the validation failure.

peReason

The returned Invalid reason.

pRespCode

The returned response code.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjSetRecordRouteMode()

DESCRIPTION

Indicates whether or not the specified *ProxyCoreObj* should use the Record Route mechanism. You can use this function only in the transaction stateful *ProxyCoreObj* mode.

SYNTAX

```
RvStatus RvProxyCoreObjSetRecordRouteMode(
    IN RvProxyCoreObjHandle          hCoreObj,
    IN RvSipServerRecordRouteMode    eRecordRoute);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to set this attribute.

eRecordRoute

The Record Route mode.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjSetViaHeader()

DESCRIPTION

Sets a Via header that should be added to the request before it is sent. You can use this function only in transaction stateful *ProxyCoreObj* mode.

SYNTAX

```
RvStatus RvProxyCoreObjSetViaHeader(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvSipViaHeaderHandle    hViaHeader);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to which to set the Via header.

hViaHeader

The handle to the Via header to add to the request.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjGetState()

DESCRIPTION

Returns the current state of the specified *ProxyCoreObj*.

SYNTAX

```
RvStatus RvProxyCoreObjGetState(
    IN  RvProxyCoreObjHandle    hCoreObj,
    OUT RvProxyCoreObjState     *pState);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to return its state.

pState

The returned *ProxyCoreObj* state.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjGetMode()

DESCRIPTION

Returns the *ProxyCoreObj* working mode.

SYNTAX

```
RvStatus RvProxyCoreObjGetMode(
    IN  RvProxyCoreObjHandle    hCoreObj,
    OUT RvSipSeverMode          *pMode);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to return its mode.

pMode

The returned mode.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjGetPolicyMgr()

DESCRIPTION

Gets the *ProxyPolicyMgr* handle "owner" of the *ProxyCoreObj*.

SYNTAX

```
RvStatus RvProxyCoreObjGetPolicyMgr(
    IN  RvProxyCoreObjHandle       hCoreObj,
    OUT RvProxyPolicyMgrHandle*    phPolicyMgr);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* handle.

phPolicyMgr

The handle of the *ProxyPolicyMgr*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjGetReceivedLocalAddress()

DESCRIPTION

Gets the local address on which the request was received. This function is valid in the following states:

- RVPROXY_CORE_OBJ_STATE_REQUEST_RECEIVED
- RVPROXY_CORE_OBJ_STATE_INVALID_REQUEST
- RVPROXY_CORE_OBJ_STATE_AUTHENTICATING
- RVPROXY_CORE_OBJ_STATE_AUTHENTICATION_FAILED
- RVPROXY_CORE_OBJ_STATE_REQUEST_VALID
- RVPROXY_CORE_OBJ_STATE_RESOLVING_DESTINATION
- RVPROXY_CORE_OBJ_STATE_ADDRESS_RESOLVED

SYNTAX

```
RvStatus RvProxyCoreObjGetReceivedLocalAddress(
    IN  RvProxyCoreObjHandle          hCoreObj,
    OUT RvSipTransport*               peTransportType,
    OUT RvSipTransportAddressType*    peAddressType,
    OUT RvChar*                       pszLocalAddress,
    OUT RvUint16*                     pLocalPort);
```

PARAMETERS

hCoreObj

The handle to the *ProxyCoreObj* to return its received local address.

peTransportType

The transport type (UDP or TCP).

peAddressType

The address type (IPv4 or IPv6).

pszLocalAddress

The local address.

pLocalPort

The local port.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjAddRoutesToRequest()

DESCRIPTION

Adds Route headers to an outgoing request. This function should be called before calling RvProxyCoreObjProxyRequest(). The Route headers will be added to each of the outgoing requests (in the case that more than one request will be generated when forwarding the original request).

All Route headers that are added should be of proxies that are loose routers (with the *lr* parameter). The head of the list will be set as the Top Route header in the outgoing request. Using this function implies that the proxy has a local policy that mandates that a request visit a specific set of proxies before being delivered to the destination.

SYNTAX

```
RvStatus RvProxyCoreObjAddRoutesToRequest(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvProxyListHandle       hRouteList);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* object to which to add the Route headers.

hRouteList

A list of Route headers to add to the forward request. The Route must be of proxies that are loose routers. The Routes will be added according to the order supplied in the header list (the first header will be the top most Route in the outgoing message).

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjSetSTPreferenceParams()

DESCRIPTION

Sets the Session Timer preference parameters associated with this *ProxyCoreObj*. If the input *minSE* parameter equals UNDEFINED, a default value according to the standard (90) is set. This function is valid only for a stateful *ProxyCoreObj* in a pre-forwarding state.

SYNTAX

```
RvStatus RvProxyCoreObjSetSTPreferenceParams(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvInt32                 minSE,
    IN RvInt32                 sessionExpires);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to set its Session Timer preference parameters.

minSE

The Min-SE interval to use.

sessionExpires

The Session-Expires interval to use.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjAddAuthHeaderToResponse()

DESCRIPTION

Indicates whether or not the specified *ProxyCoreObj* should add a WWW-Authentication header or Proxy-Authentication header to an initiated 401 or 407 response message, respectively.

SYNTAX

```
RvStatus RvProxyCoreObjAddAuthHeaderToResponse(
    IN RvProxyCoreObjHandle    hCoreObj,
    IN RvBool                  bAddAuthHeader);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to set this attribute.

bAddAuthHeader

If RV_TRUE, adds the Un-Authenticated header to the 401 or 407 response message. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjServerTranscSetTimers()

DESCRIPTION

Sets new timer values to the *ProxyCoreObj* server transaction. According to RFC 3261, the transaction has to set different timers during its life cycle and perform different actions when the timers expire. For example, after sending a final response, the transaction has to set a timer to the value of 32,000 MSec. When this timer expires the transaction must terminate. The values of the transaction timers are taken from the server configuration and are defined upon initialization. The application can use the RvProxyCoreObjServerTranscSetTimers() API function to change the different timer values of the *ProxyCoreObj* server transaction. The RvSipTimers structure received by this function contains all the available configurable timers. You can set values to the timers that you want to configure. You can set UNDEFINED to timers you wish the Stack to calculate. If you set a timer value to UNDEFINED, and this timer cannot be calculated from other timers in the structure, the timer value will be taken from the server configuration. The RvSipTimers structure also enables you to change the number of retransmissions performed by the transaction. The default number of retransmissions is calculated using the different transaction timers. However, the application can set a different number and change the retransmission count.

SYNTAX

```
RvProxyCoreObjServerTranscSetTimers(
    IN  RvProxyCoreObjHandle    hCoreObj,
    IN  RvSipTimers*            pTimers,
    IN  RvInt32                 sizeOfStruct)
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* to set this attribute.

pTimers

A ointer to the structure that contains all the timeout values.

sizeOfStruct

The size of the RvSipTimers structure.

RETURN VALUES

Returns RvStatus.

PROXY CORE TRANSACTION FUNCTIONS

The Proxy Core Transaction API functions enable you to control a proxy transaction and access the Proxy Core Transaction parameters. These functions are included in the *RvProxyTransaction.h* header file.

The Proxy Core Transaction API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvProxyCoreTranscTerminate()
- RvProxyCoreTranscCancel()
- RvProxyCoreTranscDnsContinue()
- RvProxyCoreTranscDnsTerminate()
- RvProxyCoreTranscMake()
- RvProxyCoreTranscRequest()
- RvProxyCoreTranscRequestMsg()
- RvProxyCoreTranscSendToFirstRoute()

RvProxyCoreTranscTerminate()

DESCRIPTION

Terminates the specified *ProxyCoreTransc*.

SYNTAX

```
RvStatus RvProxyCoreTranscTerminate(
    IN RvProxyTranscHandle    hProxyTransc);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* to terminate.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscCancel()

DESCRIPTION

Sends CANCEL on a given pending *ProxyCoreTransc*.

SYNTAX

```
RvStatus RvProxyCoreTranscCancel(
    IN RvProxyTranscHandle    hProxyTransc,
    IN RvBool                 bCheckTranscProceeding);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* to cancel.

bCheckTranscProceeding

Indicates whether or not to check if the transaction is proceeding (received 1xx) before canceling.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscDnsContinue()

DESCRIPTION

Continues sending the request to the next address in the DNS address list. This function may be called after the application received notification about a sending failure in the RvProxyCoreTranscStateChangeEv() event with the MSG_SEND_FAILURE state, and wishes to continue sending the request to the next address in the DNS list. Before calling this function, the application can manipulate the DNS address list. The request will be sent to the first address in the list.

SYNTAX

```
RvStatus RvProxyCoreTranscDnsContinue(
    IN RvProxyTranscHandle    hProxyTransc);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* to cancel.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscDnsTerminate()

DESCRIPTION

Does not continue to next address in the DNS address list. Terminates the current *ProxyCoreTransc* (and the SIP Stack transaction that was already cloned at EvSendFailure). In case of sequential proxying, continues to the next address in the destAddress list.

SYNTAX

```
RvStatus RvProxyCoreTranscDnsTerminate(
    IN RvProxyTranscHandle    hProxyTransc);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* of which to stop continuing to its next DNS address.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscMake()

DESCRIPTION

Initializes a *ProxyCoreTransc* and sends a request. Before calling this function you should call RvProxyCorePolicyMgrCreateSFCoreObj() to create a new *ProxyCoreObj* and *ProxyCoreTransc*. The SIP Server Platform will automatically create a call-ID. You can use RvProxyCoreTranscSetCallId() before calling this function to specify a different call-ID.

SYNTAX

```
RvStatus RvProxyCoreTranscMake(
    IN RvProxyTranscHandle    hProxyTransc,
    IN RvChar*                strFrom,
    IN RvChar*                strTo,
    IN RvChar*                strRequestURI,
    IN RvInt32                cseqStep,
    IN RvChar*                strMethod);
```

PARAMETERS

hProxyTransc

The handle of the *ProxyCoreTransc* that sends the request.

strFrom

A NULL terminated string with the From header to set.

strTo

A NULL terminated string with the To header to set.

strRequestURI

A NULL terminated string with the Request URI.

cseqStep

The CSeq-Step number to set.

strMethod

A NULL terminated string with name of the method to send.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscRequest()

DESCRIPTION

Creates and sends a request message according to the given method, with the given address as a Request URI. Call this function after creating a *ProxyCoreObj* and *ProxyCoreTransc* using RvProxyCorePolicyMgrCreateSFCoreObj(), and setting all relevant identifiers to it.

SYNTAX

```
RvStatus RvProxyCoreTranscRequest(
    IN RvProxyTranscHandle    hProxyTransc,
    IN RvChar*                strRequestMethod,
    IN RvSipAddressHandle     hRequestUri);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* to which this request refers.

strRequestMethod

The request method string.

pRequestUri

The Request URI to be used in the request message. The Request URI of the message will be the address given in this parameter, and the message will be sent accordingly.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscRequestMsg()

DESCRIPTION

Sends a request message to the remote party using a given prepared message object. This function will normally be used when the message to be sent is already prepared. The user is responsible for setting the correct Request URI in the given message object and for applying the Record-Route rules if necessary. Call this function after creating a *ProxyCoreObj* and *ProxyCoreTransc* using RvProxyCorePolicyMgrCreateSFCoreObj().

SYNTAX

```
RvStatus RvProxyCoreTranscRequestMsg(
    IN RvProxyTranscHandle    hProxyTransc,
    IN RvSipMsgHandle         hMsg,
    IN RvBool                 bAddTopVia);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* to which this request refers.

hMsg

The handle of the request message object. This message will be sent to the remote party.

bAddTopVia

Indicates whether or not the SIP Server Platform should add a top Via header to the request message before the message is sent. You should set this parameter to RV_TRUE if you wish the SIP Server Platform to handle the Via header for you. If you have added the Via header by yourself, set this parameter to RV_FALSE.

Note If you wish the SIP Server Platform to add a specific branch parameter to the top Via header, you should first use the RvProxyCoreTranscSetViaBranch() function and set the branch in the transaction.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSendToFirstRoute()

DESCRIPTION

Indicates whether to send the request according to first Route header (as a loose router) or according to the Request-Uri. If not set, the default is to send the request according to the Request-Uri.

SYNTAX

```
RvProxyCoreTranscSendToFirstRoute(
    IN  RvProxyTranscHandle    hProxyTransc,
    IN  RvBool                 bSendToFirstRouteHeader)
```

PARAMETERS

hProxyTransc

The *transaction* in which to set the *bSendToFirstRouteHeader*.

bSendToFirstRouteHeader

The bSendToFirstRouteHeader value to set in the *transactio*n.

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

The Get and Set functions are:

- RvProxyCoreTranscGetCancelPair()
- RvProxyCoreTranscGetProxyCoreObj()
- RvProxyCoreTranscGetReceivedResponse()
- RvProxyCoreTranscGetResponseInvalidReason()
- RvProxyCoreTranscGetDnsAddrList()
- RvProxyCoreTranscGetState()
- RvProxyCoreTranscGetConnection()
- RvProxyCoreTranscGetPersistency()
- RvProxyCoreTranscSetConnection()
- RvProxyCoreTranscSetPersistency()
- RvProxyCoreTranscGetLocalAddress()
- RvProxyCoreTranscSetLocalAddress()
- RvProxyCoreTranscGetResolvedFinalDest()
- RvProxyCoreTranscSetFromHeader()
- RvProxyCoreTranscGetFromHeader()
- RvProxyCoreTranscSetToHeader()
- RvProxyCoreTranscGetToHeader()
- RvProxyCoreTranscSetCallId()
- RvProxyCoreTranscGetCallId()
- RvProxyCoreTranscSetCSeqStep()
- RvProxyCoreTranscGetCSeqStep()
- RvProxyCoreTranscSetViaBranch()
- RvProxyCoreTranscGetOutboundMsg()
- RvProxyCoreTranscSetOutboundAddr()
- RvProxyCoreTranscGetOutboundAddr()
- RvProxyCoreTranscGetRequestUri()
- RvProxyCoreTranscSendToFirstRoute()
- RvProxyCoreTranscGetStackTransmitter()
- RvProxyCoreTranscSetForceOutboundAddrFlag()
- RvProxyCoreTranscSetTimers()

RvProxyCoreTranscGetCancelPair()

DESCRIPTION

Returns the *ProxyCoreTransc* cancel pair. This cancel pair is the internally created *ProxyCoreObj* which was created when the application called RvProxyCoreTranscCancel(). This *ProxyCoreObj* is the outgoing CANCEL request. If a cancel pair was not found, hCancelPair will be NULL.

SYNTAX

```
RvStatus RvProxyCoreTranscGetCancelPair(
    IN  RvProxyTranscHandle     hProxyTransc,
    OUT RvProxyCoreObjHandle    *phCoreObj);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* to return its cancel pair.

phCoreObj

The returned cancel *ProxyCoreObj* pair.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetProxyCoreObj()

DESCRIPTION

Returns the *ProxyCoreObj* that "owns" the given *ProxyCoreTransc*.

SYNTAX

```
RvStatus RvProxyCoreTranscGetProxyCoreObj(
    IN  RvProxyTranscHandle        hTransac,
    OUT RvProxyCoreObjHandle       *phCoreObj,
    OUT RvProxyCoreObjAppHandle    *phCoreObjAppHandle);
```

PARAMETERS

hTransac

The proxy transaction whose *ProxyCoreObj* "owner" should be returned.

phCoreObj

The returned *ProxyCoreObj*.

phCoreObjAppHandle

The returned application *ProxyCoreObj*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetReceivedResponse()

DESCRIPTION

Returns a handle to the last response message that the *ProxyCoreTransc* object received. This function succeeds even if no response message has been found, therefore phMsg must be inspected before further usage.

SYNTAX

```
RvStatus RvProxyCoreTranscGetReceivedResponse(
    IN  RvProxyTranscHandle    hProxyTransc,
    OUT RvSipMsgHandle         *phMsg);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* object whose last received response should be returned.

phMsg

The returned handle of the received response.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetResponseInvalidReason()

DESCRIPTION

Gets the reason for the response validation failure. If the response is valid and this function is called, the function will return UNDEFINED on *peReason*.

SYNTAX

```
RvStatus RvProxyCoreTranscGetResponseInvalidReason(
    IN  RvProxyTranscHandle        hTransac,
    OUT RvProxyCoreTranscReason    *phMsg);
```

PARAMETERS

hTransac

The *ProxyCoreTransc* that received the invalid response.`RvProxyCoreTranscReason`

peReason

The returned invalid response reason.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetDnsAddrList()

DESCRIPTION

Returns a transport DNS list object to which a *ProxyCoreTransc* will send the message in case of a sending failure. This list will be valid only after the first sending attempt. The function returns a pointer to an internal object. The application must not free this list but can change its order. This function is supported only in transaction stateful mode.

SYNTAX

```
RvStatus RvProxyCoreTranscGetDnsAddrList(
    IN  RvProxyTranscHandle             hTransac,
    OUT RvSipTransportDNSListHandle*    phDnsList);
```

PARAMETERS

hTransac

The *ProxyCoreTransc*.

phDnsList

The returned DNS list.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetState()

DESCRIPTION

Returns the current state of the specified *ProxyCoreTransc*.

SYNTAX

```
RvStatus RvProxyCoreTranscGetState(
    IN  RvProxyTranscHandle       hTransac,
    OUT RvProxyCoreTranscState    *pState)
```

PARAMETERS

hTransac

The *ProxyCoreTransc* whose state is to be returned.

pState

A pointer to the variable to set the state.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetConnection()

DESCRIPTION

Returns the connection that the *ProxyTransc* is currently using.

Note Only persistent *ProxyTransc* keep the currently-used connection.

SYNTAX

```
RvStatus RvProxyCoreTranscGetConnection(
    IN  RvProxyTranscHandle                hTransac,
    OUT RvSipTransportConnectionHandle*    phConn);
```

PARAMETERS

hTransac

The *ProxyTransc* object.

phConn

The handle of the connection that the *ProxyTransc* is currently using.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetPersistency()

DESCRIPTION

Returns whether or not the *ProxyTransc* is configured to try using a persistent connection.

SYNTAX

```
RvStatus RvProxyCoreTranscGetPersistency(
    IN  RvProxyTranscHandle    hTransac,
    OUT RvBool*                pbIsPersistent);
```

PARAMETERS

hTransac

The *ProxyTransc* object.

pbIsPersistent

Determines whether or not the *ProxyTransc* uses a persistent connection.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSetConnection()

DESCRIPTION

Sets a connection to be used by the *ProxyTransc*. The *ProxyTransc* will try to use the given connection for outgoing requests only. Responses are sent on the same connection as the request. If the connection will not fit the *ProxyTransc* local and remote addresses, it will be replaced. You can set the transaction connection only on the RvProxyCoreTranscCreatedEv() callback (before the request is sent).

Note This function can be used only if the *ProxyTransc* is persistent.

SYNTAX

```
RvStatus RvProxyCoreTranscSetConnection(
    IN RvProxyTranscHandle              hTransac,
    IN RvSipTransportConnectionHandle   hConn);
```

PARAMETERS

hTransac

The *ProxyTransc* object.

hConn

The handle to the connection.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSetPersistency()

DESCRIPTION

Instructs the *ProxyTransc* on whether or not to try using a persistent connection. You can change the *ProxyTransc* persistency only at the RvProxyCoreTranscCreatedEv() callback (before request is sent).

SYNTAX

```
RvStatus RvProxyCoreTranscSetPersistency(
    IN RvProxyTranscHandle    hTransac,
    IN RvBool                 pbIsPersistent);
```

PARAMETERS

hTransac

The *ProxyTransc* object.

bIsPersistent

Determines whether or not the transaction will try to use a persistent connection.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetLocalAddress()

DESCRIPTION

Gets the local address of the *ProxyCoreTransc* that will be used to send outgoing requests. This is the address that is set using the RvProxyCoreTranscSetLocalAddress() function. If no address was set, this function will return the first UDP address. Users can use the transport and address types to indicate which kind of local address they wish to get.

Note This function cannot be used for a *ProxyCoreTransc* whose *ProxyCoreObj* owner ID is a stateless *ProxyCoreObj*.

SYNTAX

```
RvStatus RvProxyCoreTranscGetLocalAddress(
    IN  RvProxyTranscHandle          hProxyTrans,
    IN  RvSipTransport               eTransportType,
    IN  RvSipTransportAddressType    eAddressType,
    OUT RvChar*                      szLocalAddress,
    OUT RvUint16*                    pLocalPort);
```

PARAMETERS

hProxyTrans

The handle to the *ProxyCoreTransc* to get its local address.

eTransportType

The transport type (UDP, TCP, SCTP or TLS).

eAddressType

The address type (IPv4 or IPv6).

szLocalAddress

The local address that this *ProxyCoreTransc* is using.

pLocalPort

The local port that this *ProxyCoreTransc* is using.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSetLocalAddress()

DESCRIPTION

Sets the local address from which the *ProxyCoreTransc* will send outgoing requests. The SIP Server can be configured to listen to many local addresses. Each local address has a transport type (UDP, TCP or TLS) and an address type (IPv4 or IPv6). When the SIP Server sends an outgoing request, the local address (from where the request is sent) is chosen according to the characteristics of the remote address. Both the local and remote addresses must have the same characteristics, such as the same transport and address types. If several local addresses are configured, the SIP Server will use the first match that was configured.

You can use the RvSipProxyCoreSetLocalAddress() function to force the *ProxyCoreTransc* to choose a specific local address for a specific transport and address type. For example, you can force the *ProxyCoreTransc* to use the second configured UDP/IPv4 local address. If the *ProxyCoreTransc* sends a request to a UDP/IPv4 remote address, it will use the local address that you set instead of the default first local address.

Note
**The localAddress string you provide for this function must match exactly with the local address that was inserted in the configuration structure at the initialization of the SIP Server.
**You cannot use the 0.0.0.0 notation when using the multi-homed feature.
**You cannot use this function for a *ProxyCoreTransc* whose *ProxyCoreObj* owner ID is a stateless *ProxyCoreObj*.

SYNTAX

```
RvStatus RvProxyCoreTranscSetLocalAddress(
    IN RvProxyTranscHandle          hProxyTrans,
    IN RvSipTransport               eTransportType,
    IN RvSipTransportAddressType    eAddressType,
    IN RvChar*                      szLocalAddress,
    IN RvUint16                     localPort);
```

PARAMETERS

hProxyTrans

The handle to the *ProxyCoreTransc* to set its local address.

eTransportType

The transport type (UDP, TCP, SCTP or TLS) to set to the *ProxyCoreTransc*.

eAddressType

The address type (IPv4 or IPv6) to set to the *ProxyCoreTransc*.

szLocalAddress

The local address to be set to this *ProxyCoreTransc*.

localPort

The local port to be set to this *ProxyCoreTransc*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetResolvedFinalDest()

DESCRIPTION

Returns the final destination (IP, port, transport and address type) that the *ProxyCoreTransc* will send the request to. This function is valid only in the RvProxyCoreObjRequestFinalDestResolvedEv() callback.

SYNTAX

```
RvStatus RvProxyCoreTranscGetResolvedFinalDest(
    IN  RvProxyTranscHandle           hProxyTrans,
    OUT RvSipTransport*               peTransportType,
    OUT RvSipTransportAddressType*    peAddressType,
    OUT RvChar*                       szDestAddr,
    OUT RvUint16*                     pDestPort);
```

PARAMETERS

hProxyTrans

The handle to the *ProxyCoreTransc* to get its request final destination address.

eTransportType

The returned destination transport type (UDP, TCP, SCTP or TLS).

eAddressType

The returned destination address type (IPv4 or IPv6).

szDestAddr

The returned destination address of the request.

pDestPort

The returned destination port of the request.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSetFromHeader()

DESCRIPTION

Sets the From header of the *ProxyCoreTransc*.

SYNTAX

```
RvStatus RvProxyCoreTranscSetFromHeader(
    IN RvProxyTranscHandle       hProxyTransc,
    IN RvSipPartyHeaderHandle    hFrom);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* from which to set the From header.

hFrom

A pointer to the From header handle to set to the *ProxyCoreTransc*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetFromHeader()

DESCRIPTION

Returns the From header of the *ProxyCoreTransc*.

SYNTAX

```
RvStatus RvProxyCoreTranscGetFromHeader(
    IN  RvProxyTranscHandle        hProxyTransc,
    OUT RvSipPartyHeaderHandle*    phFrom);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* from which to get the From header.

phFrom

A pointer to the From header handle of the *ProxyCoreTransc.*

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSetToHeader()

DESCRIPTION

Sets the To header of the *ProxyCoreTransc*.

SYNTAX

```
RvStatus RvProxyCoreTranscSetToHeader(
    IN RvProxyTranscHandle       hProxyTransc,
    IN RvSipPartyHeaderHandle    hTo);
```

PARAMETERS

hProxyTransc

The transaction from which to set the To header.

hTo

A pointer to the To header handle to set to the *ProxyCoreTransc*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetToHeader()

DESCRIPTION

Returns the To address associated with the *ProxyCoreTransc*. Attempting to alter the To address after transaction left the Initial state might cause unexpected results.

SYNTAX

```
RvStatus RvProxyCoreTranscGetToHeader(
    IN  RvProxyTranscHandle        hProxyTransc,
    OUT RvSipPartyHeaderHandle*    phTo);
```

PARAMETERS

hProxyTransc

The handle to the *ProxyCoreTransc*.

phTo

A pointer to the To header handle of the *ProxyCoreTransc*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSetCallId()

DESCRIPTION

Sets the call-ID value of the *ProxyCoreTransc*.

SYNTAX

```
RvStatus RvProxyCoreTranscSetCallId(
    IN RvProxyTranscHandle    hProxyTransc,
    IN RvChar*                strCallId);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* to set the call-ID value.

strCallId

The call ID string. Must be NULL terminated.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetCallId()

DESCRIPTION

Returns the call-ID value of the *ProxyCoreTransc*. The call-ID is copied to the given buffer.

SYNTAX

```
RvStatus RvProxyCoreTranscGetCallId(
    IN    RvProxyTranscHandle    hProxyTransc,
    OUT   RvChar*                strCallId,
    INOUT RvInt32*               len);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* from which to get the call-ID value.

strCallId

The string with the call-ID (in case the buffer length was sufficient).

len

IN: The size of the allocated application buffer.

OUT: The actual size of the call-ID.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSetCSeqStep()

DESCRIPTION

Sets the CSeq-Step number of the *ProxyCoreTransc*.

SYNTAX

```
RvStatus RvProxyCoreTranscSetCSeqStep(
    IN RvProxyTranscHandle    hProxyTransc,
    IN RvInt32                cseqStep);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* to which to set the CSeq-Step value.

cseqStep

The value of the CSeq-Step to be set.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetCSeqStep()

DESCRIPTION

Gets the CSeq-Step number of the *hProxyTransc*.

SYNTAX

```
RvStatus RvProxyCoreTranscGetCSeqStep(
    IN  RvProxyTranscHandle    hProxyTransc,
    OUT RvInt32*               pCSeqStep);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* from which to get the CSeq-Step value.

pCSeqStep

The CSeq-Step number of the *ProxyCoreTransc*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSetViaBranch()

DESCRIPTION

Sets the top Via branch of the *ProxyCoreTransc*. The *ProxyCoreTransc* will add the branch to the top Via header of outgoing requests.

SYNTAX

```
RvStatus RvProxyCoreTranscSetViaBranch(
    IN RvProxyTranscHandle    hProxyTransc,
    IN RvChar*                strBranch);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* handle.

strBranch

The NULL terminated string with the Via branch to set.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetOutboundMsg()

DESCRIPTION

Gets the message that the *ProxyCoreTransc* is going to send. This function is only valid in states where the *ProxyCoreTransc* is about to send a request message, such as RVPROXY_CORE_TRANSC_STATE_IDLE. You can call this function before you call RvProxyCoreTranscRequest() to add information to the outgoing request.

Note The message you receive from this function is not complete. In some cases it might even be empty. You should use this function to add headers to the message before it is sent. To view the complete message, use the RvProxyCoreObjMsgToSendEv() callback function.

SYNTAX

```
RvStatus RvProxyCoreTranscGetOutboundMsg(
    IN  RvProxyTranscHandle    hProxyTransc,
    OUT RvSipMsgHandle*        phMsg);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* handle.

phMsg

A pointer to the message.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSetOutboundAddr()

DESCRIPTION

Sets all outbound proxy details to the *ProxyCoreTransc*. All details are supplied in the RvSipTransportOutboundAddress structure that includes parameters such as the IP address, host name, transport and port. Requests that this object sent will use the outbound detail specifications as a remote address. The Request-URI will be ignored. If the object has a Route Set, the outbound details will be ignored.

Note If you specify both IP address and a host name in the Configuration structure, either of them will be set **but** the IP address is preferably used. If you do not specify port or transport, both will be determined according to the DNS procedures specified in RFC 3263

SYNTAX

```
RvStatus RvProxyCoreTranscSetOutboundAddr(
    IN RvProxyTranscHandle                hProxyTransc,
    IN RvSipTransportOutboundProxyCfg     *pOutboundCfg,
    IN RvInt32                            sizeOfCfg);
```

PARAMETERS

hProxyTransc

The transaction handle.

pOutboundCfg

A pointer to the outbound address configuration structure with all the relevant details.

sizeOfCfg

The size of the outbound address configuration structure.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetOutboundAddr()

DESCRIPTION

Gets all outbound proxy details that the *ProxyCoreTransc* uses. The details are placed in the RvSipTransportOutboundAddress structure that includes parameters such as the IP address, host name, transport and port. If the outbound details were not set to the specific *ProxyCoreTransc*, but an outbound proxy was defined to the SIP Server on initialization, the SIP Server parameters will be returned. If the *ProxyCoreTransc* does not use an outbound address, NULL/ UNDEFINED values are returned.

Note You must supply a valid consecutive buffer in the RvSipTransportOutboundAddress structure to get the outbound strings (host name and IP address).

SYNTAX

```
RvStatus RvProxyCoreTranscGetOutboundAddr(
    IN  RvProxyTranscHandle              hProxyTransc,
    IN  RvInt32                          sizeOfCfg,
    OUT RvSipTransportOutboundProxyCfg   *pOutboundCfg);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* handle.

sizeOfCfg

The size of the configuration structure.

pOutboundCfg

A pointer to outbound address configuration structure.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetRequestUri()

DESCRIPTION

Returns the Request URI that the transaction is using.

SYNTAX

```
RvStatus RvProxyCoreTranscGetRequestUri(
    IN  RvProxyTranscHandle    hProxyTransc,
    OUT RvSipAddressHandle     *hReqUri);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* to return its Request URI.

hReqUri

The Request URI.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSendToFirstRoute()

DESCRIPTION

Indicates whether to send the request according to first Route header (as a loose router) or according to Request URI. If not set, the default is to send the request according to Request URI.

SYNTAX

```
RvStatus RvProxyCoreTranscSendToFirstRoute(
    IN RvProxyTranscHandle    hProxyTransc,
    IN RvBool                 bSendToFirstRouteHeader);
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* in which to set the *bSendToFirstRouteHeader*.

bSendToFirstRouteHeader

The *bSendToFirstRouteHeader* value to set in the *ProxyCoreTransc*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscGetStackTransmitter()

DESCRIPTION

Returns a handle to the underlaying SIP Stack *transmitter* object.

SYNTAX

```
RvProxyCoreTranscGetStackTransmitter(
    IN  RvProxyTranscHandle        hProxyTransc,
    OUT RvSipTransmitterHandle*    phTransmitter)
```

PARAMETERS

hProxyTransc

The handle of the *ProxyCoreTransc*.

phTransmitter

The returned SIP Stack *transmitter* handle.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSetForceOutboundAddrFlag()

DESCRIPTION

Sets the “force outbound addr” flag. This flag forces the *ProxyCoreTransc* to send the request to the outbound address regardless of the message content.

SYNTAX

```
RvProxyCoreTranscSetForceOutboundAddrFlag(
    IN  RvProxyTranscHandle    hProxyTransc,
    IN  RV_BOOL                bForceOutboundAddr)
```

PARAMETERS

hProxyTransc

The handle of the *ProxyCoreTransc*.

bForceOutboundAddr

The flag value to set.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTranscSetTimers()

DESCRIPTION

Sets new timer values to the *ProxyCoreTransc*. According to RFC 3261, the transaction has to set different timers during its life cycle and perform different actions when the timers expire. For example, after sending a final response, the transaction has to set a timer to the value of 32,000 MSec. When this timer expires the transaction must terminate. The values of the *ProxyCoreTransc* timers are taken from the server configuration and are defined upon initialization. The application can use the RvProxyCoreTranscSetTimers() API function to change the different timer values of the ProxyTransc. The RvSipTimers structure received by this function contains all the available configurable timers. You can set values to the timers you want to configure. You can set UNDEFINED to timers you wish the stack to calculate. If you set a timer value to UNDEFINED, and this timer cannot be calculated from other timers in the structure, the timer value will be taken from the server configuration. The RvSipTimers structure also enables you to change the number of retransmissions performed by the transaction. The default number of retransmissions is calculated using the different transaction timers. However, the application can set a different number and change the retransmission count.

SYNTAX

```
RvProxyCoreTranscSetTimers(
    IN  RvProxyTranscHandle    hProxyTransc,
    IN  RvSipTimers*           pTimers,
    IN  RvInt32                sizeOfStruct)
```

PARAMETERS

hProxyTransc

The *ProxyCoreTransc* handle.

pTimers

Aointer to the structure that contains all the timeout values.

sizeOfStruct

The size of the RvSipTimers structure.

RETURN VALUES

Returns RvStatus.

PROXY CORE TRANSMITTER FUNCTIONS

The Proxy Core Transmitter API functions enable you to control a *ProxyTrx* and access the Proxy Core Transmitter parameters. These functions are included in the *RvProxyTrx.h* header file.

The Proxy Core Transmitter API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvProxyCoreTrxSendMsg()
- RvProxyCoreTrxTerminate()

RvProxyCoreTrxSendMsg()

DESCRIPTION

Sends a message to a remote party using a given prepared message object. This function is normally used when the message to be sent is already prepared. The user is responsible for setting the correct Request-URI in the given message object and for applying the Record-Route rules if necessary. Call this function after creating a *ProxyCoreObj* and *ProxyTrx* using the RvProxyCorePolicyMgrCreateSLCoreObj() function.

Note When working with DNS re-sending of a message (except for ACK msg) results in switching to stateful mode (RFC 3263).

SYNTAX

```
RvStatus RvProxyCoreTrxSendMsg(
    IN RvProxyTrxHandle    hProxyTrx,
    IN RvSipMsgHandle      hMsgToSend,
    IN RvBool              bAddTopVia);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* this message refers to.

hMsgToSend

The handle of the message object. This message will be sent to the remote party. This parameter is ignored if the current state is MSG_SEND_FAILURE (except for an ACK message). In the MSG_SEND_FAILURE state, the original message will be re-processed and sent. This parameter may be NULL for an ACK message if the current state is MSG_SEND_FAILURE, and KeepMsgFlag was set to RV_TRUE.

bAddTopVia

Indicates whether or not the SIP Server should add a top Via header to the request message before the message is sent. You should set this parameter to RV_TRUE if you want the SIP Server to handle the Via header for you. If you added the Via header yourself, set this parameter to RV_FALSE.

Note If you want the SIP Server to add a branch parameter to the top Via header, you should first use the RvProxyCoreTrxSetViaBranch() function and set the branch in the *ProxyTrx*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxTerminate()

DESCRIPTION

Terminates the specified *ProxyTrx*.

SYNTAX

```
RvStatus RvProxyCoreTrxTerminate(
    IN RvProxyTrxHandle    hProxyTrx);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* to terminate.

RETURN VALUES

Returns RvStatus.

Get and Set Functions

The Get and Set functions are:

- RvProxyCoreTrxGetConnection()
- RvProxyCoreTrxGetCurrentLocalAddress()
- RvProxyCoreTrxGetDestAddress()
- RvProxyCoreTrxGetDnsAddrList()
- RvProxyCoreTrxGetLocalAddress()
- RvProxyCoreTrxGetOutboundAddr()
- RvProxyCoreTrxGetPersistency()
- RvProxyCoreTrxGetProxyCoreObj()
- RvProxyCoreTrxGetState()
- RvProxyCoreTrxSetConnection()
- RvProxyCoreTrxSetDestAddress()
- RvProxyCoreTrxSetIgnoreOutboundProxyFlag()
- RvProxyCoreTrxSetKeepMsgFlag()
- RvProxyCoreTrxSetLocalAddress()
- RvProxyCoreTrxSetOutboundAddr()
- RvProxyCoreTrxSetPersistency()
- RvProxyCoreTrxSetViaBranch()
- RvProxyCoreTrxSetForceOutboundAddrFlag()

RvProxyCoreTrxGetConnection()

DESCRIPTION

Returns the connection that the *ProxyTrx* is currently using.

SYNTAX

```
RvStatus RvProxyCoreTrxGetConnection(
    IN  RvProxyTrxHandle                    hProxyTrx,
    OUT RvSipTransportConnectionHandle*     phConn);
```

PARAMETERS

hProxyTrx

The *ProxyTrx*.

phConn

The handle of the connection that the *ProxyTrx* is currently using.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxGetCurrentLocalAddress()

DESCRIPTION

Gets the local address the *ProxyTrx* will use to send the next message. This function returns the actual address from the six addresses that were used, or are going to be used.

SYNTAX

```
RvStatus RvProxyCoreTrxGetCurrentLocalAddress(
    IN  RvProxyTrxHandle              hProxyTrx,
    OUT RvSipTransport*               eTransportType,
    OUT RvSipTransportAddressType*    eAddressType,
    OUT RvChar*                       localAddress,
    OUT RvUint16*                     localPort);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* to get its current local address.

eTransportType

The transport type (UDP, TCP, SCTP or TLS).

eAddressType

The address type (IPv4 or IPv6).

localAddress

The local address this *ProxyTrx* is using. (The address must be longer than 48).

localPort

The local port this *ProxyTrx* is using.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxGetDestAddress()

DESCRIPTION

Returns the destination address that the *ProxyTrx* will use.

SYNTAX

```
RvStatus RvProxyCoreTrxGetDestAddress(
    IN  RvProxyTrxHandle               hProxyTrx,
    IN  RvInt32                        addrStructSize,
    IN  RvInt32                        optionsStructSize,
    OUT RvSipTransportAddr*            pDestAddr,
    OUT RvSipTransmitterExtOptions*    pOptions);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* to get its destination address.

addrStructSize

The size of the *pDestAddr* structure.

optionsStructSize

The size of the *pOptions* structure.

pDestAddr

The destination address that the *ProxyTrx* will use.

pOptions

Advanced instructions for the message sending, such as using compression.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxGetDnsAddrList()

DESCRIPTION

Returns a transport DNS list object to which a *ProxyTrx* will send the message in case of sending failure. The function return a pointer to an internal object. The application must not free this list, but can change its order.

SYNTAX

```
RvStatus RvProxyCoreTrxGetDnsAddrList(
    IN  RvProxyTrxHandle                hProxyTrx,
    OUT RvSipTransportDNSListHandle    *phDnsList);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* object.

phDnsList

The returned DNS list.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxGetLocalAddress()

DESCRIPTION

Gets the local address that the *ProxyTrx* will use to send outgoing requests. This is the address that you set using the RvProxyCoreTrxSetLocalAddress() function. If no address was set, the function will return the first UDP address. Users can use the transport type and the address type to indicate which kind of local address they wish to get.

SYNTAX

```
RvStatus RvProxyCoreTrxGetLocalAddress(
    IN  RvProxyTrxHandle             hProxyTrx,
    IN  RvSipTransport               eTransportType,
    IN  RvSipTransportAddressType    eAddressType,
    OUT RvChar*                      szLocalAddress,
    OUT RvUint16*                    pLocalPort);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* object to get its local address.

eTransportType

The transport type (UDP, TCP, SCTP or TLS).

eAddressType

The address type (IPv4 or IPv6).

szLocalAddress

The local address that this *ProxyTrx* is using.

pLocalPort

The local port that this *ProxyTrx* is using.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxGetOutboundAddr()

DESCRIPTION

Gets all outbound proxy details that the *ProxyTrx* uses. The details are placed in the RvSipTransportOutboundProxyCfg structure, which includes parameters such as the IP address or host name transport, port and compression type. If the outbound details were not set to the specific *ProxyTrx*, but the outbound proxy was defined to the SIP Server on initialization, the SIP Server parameters will be returned. If the *ProxyTrx* is not using an outbound address, NULL/ UNDEFINED values are returned.

Note You must supply a valid consecutive buffer in the RvSipTransportOutboundProxyCfg structure to get the outbound strings (host name and IP address).

SYNTAX

```
RvStatus RvProxyCoreTrxGetOutboundAddr(
    IN  RvProxyTrxHandle                    hProxyTrx,
    IN  RvInt32                             cfgStructSize,
    OUT RvSipTransportOutboundProxyCfg*     pOutboundCfg);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* to get its outbound details.

cfgStructSize

The size of the configuration structure.

pOutboundCfg

A pointer to the outbound proxy configuration structure.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxGetPersistency()

DESCRIPTION

Returns whether or not the *ProxyTrx* is configured to try using a persistent connection.

SYNTAX

```
RvStatus RvProxyCoreTrxGetPersistency(
    IN  RvProxyTrxHandle    hProxyTrx,
    OUT RvBool*             pbIsPersistent);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* object.

pbIsPersistent

Determines whether or not the *ProxyTrx* uses a persistent connection.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxGetProxyCoreObj()

DESCRIPTION

Returns the *ProxyCoreObj* that "owns" the given *ProxyTrx*.

SYNTAX

```
RvStatus RvProxyCoreTrxGetProxyCoreObj(
    IN  RvProxyTrxHandle           hProxyTrx,
    OUT RvProxyCoreObjHandle       *phCoreObj,
    OUT RvProxyCoreObjAppHandle    *phCoreObjAppHandle);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* whose *ProxyCoreObj* "owner" should be returned.

phCoreObj

The returned *ProxyCoreObj*.

phCoreObjAppHandle

The returned application *ProxyCoreObj*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxGetState()

DESCRIPTION

Returns the current state of the specified *ProxyTrx*.

SYNTAX

```
RvStatus RvProxyCoreTrxGetState(
    IN RvProxyTrxHandle         hProxyTrx,
    OUT RvProxyCoreTrxState     *pState)
```

PARAMETERS

hProxyTrx

The *ProxyTrx* whose state is to be returned.

pState

A pointer to the variable to set the state.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxSetConnection()

DESCRIPTION

Sets a connection object that the *ProxyTrx* will use. The *ProxyTrx* will hold this connection in its internal database. When sending a message, the *ProxyTrx* will use the connection only if it fits the local and remote addresses. Otherwise, the *ProxyTrx* will either locate a suitable connection in the connection hash or create a new connection. The *ProxyTrx* will inform its owner when using a different connection than the one that was supplied.

SYNTAX

```
RvStatus RvProxyCoreTrxSetConnection(
    IN RvProxyTrxHandle                 hProxyTrx,
    IN RvSipTransportConnectionHandle   hConn);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* object.

hConn

The handle to the connection.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxSetDestAddress()

DESCRIPTION

Sets a destination address that the *ProxyTrx* will use. Use this function when you want the *ProxyTrx* to use a specific address regardless of the message content. If NULL was set as the destination address, the *ProxyTrx* will continue the resolution process as if the destination IP was not received and the destination was not yet resolved.

Note Sending the *ProxyTrx* for another "round" with the DNS can take time since DNS requests and responses are exchanged.

SYNTAX

```
RvStatus RvProxyCoreTrxSetDestAddress(
    IN RvProxyTrxHandle                hProxyTrx,
    IN RvSipTransportAddr*             pDestAddr,
    IN RvInt32                         addrStructSize,
    IN RvSipTransmitterExtOptions*     pOptions,
    IN RvInt32                         optionsStructSize);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* object.

pDestAddr

The destination address the *ProxyTrx* will use. If set to NULL, the *ProxyTrx* will continue the DNS procedure as if the destination IP was never retrieved.

addrStructSize

The size of the *pDestAddr* structure.

pOptions

Advanced instructions for the message sending, such as using compression.

optionsStructSize

The size of the *pOptions* structure.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxSetIgnoreOutboundProxyFlag()

DESCRIPTION

Instructs the *ProxyTrx* to ignore its outbound proxy when sending requests. In some cases, the application will want the *ProxyTrx* to ignore its outbound proxy even if the application is configured to use one. An example is when the Request-URI was calculated from a Route header that was found in the message.

SYNTAX

```
RvStatus RvProxyCoreTrxSetIgnoreOutboundProxyFlag (
    IN RvProxyTrxHandle     hProxyTrx,
    IN RvBool               bIgnoreOutboundProxy);
```

PARAMETERS

hProxyTrx

The *ProxyTrx*.

bIgnoreOutboundProxy

RV_TRUE if you wish the *ProxyTrx* to ignore its configured outbound proxy. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxSetKeepMsgFlag()

DESCRIPTION

Indicates when the *ProxyTrx* should destruct the message, either upon termination or after encoding is completed. By default, the message will be destructed immediately after it has been encoded.

SYNTAX

```
RvStatus RvProxyCoreTrxSetKeepMsgFlag (
    IN RvProxyTrxHandle    hProxyTrx,
    IN RvBool              bKeepMsg);
```

PARAMETERS

hProxyTrx

The *ProxyTrx*.

bKeepMsg

RV_TRUE if the application wishes that the *ProxyTrx* will not destruct the message until termination. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxSetLocalAddress()

DESCRIPTION

Sets the local address from which the *ProxyTrx* will send outgoing requests. The SIP Server can be configured to listen to many local addresses. Each local address has a transport type (UDP, TCP or TLS) and an address type (IPv4or IPv6). When the SIP Server sends an outgoing request, the local address (from where the request is sent) is chosen according to the characteristics of the remote address. Both the local and remote addresses must have the same characteristics, such as the same transport type and address type. If several local addresses are configured, the SIP Server will use the first match that is configured.

You can use RvProxyCoreTrxSetLocalAddress() to force the *ProxyTrx* to choose a specific local address for a specific transport and address type. For example, you can force the *ProxyTrx* to use the second configured UDP/IPv4 local address. If the *ProxyTrx* sends a request to a UDP/IPv4 remote address, it will use the local address that you set instead of the default, first local address.

Note
** The *localAddress* string you provide for this function must match exactly with the local address that was inserted in the configuration structure at the initialization of the SIP Server.
** You cannot use the 0.0.0.0 local notation.

SYNTAX

```
RvStatus RvProxyCoreTrxSetLocalAddress (
    IN RvProxyTrxHandle             hProxyTrx,
    IN RvSipTransport               eTransportType,
    IN RvSipTransportAddressType    eAddressType,
    IN RvChar*                      szLocalAddress,
    IN RvUint16                     localPort);
```

PARAMETERS

hProxyTrx

The *ProxyTrx*.

eTransportType

The transport type (UDP, TCP, SCTP or TLS) to set to the *ProxyTrx*.

eAddressType

The address type (IPv4 or IPv6) to set to the *ProxyTrx*.

szLocalAddress

The local address to be set to this *ProxyTrx*.

localPort

The local port to be set to this *ProxyTrx*.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxSetOutboundAddr()

DESCRIPTION

Sets all outbound proxy details to the *ProxyTrx* object. All details are supplied in the RvSipTransportOutboundProxyCfg structure which includes parameters such as the IP address or host name, transport, port and compression type. Requests sent by this object will use the outbound detail specifications as a remote address. The Request-URI will be ignored. However, the outbound proxy will be ignored if the message contains a Route header or if the RvProxyCoreTrxSetIgnoreOutboundProxyFlag() function was called.

Note If you specify both an IP address and host name in the configuration structure, either of them will be set **but** the using the IP address is preferable. If you do not specify a port or transport, both will be determined according to the DNS procedures specified in RFC 3263.

SYNTAX

```
RvStatus RvProxyCoreTrxSetOutboundAddr(
    IN RvProxyTrxHandle                 hProxyTrx,
    IN RvSipTransportOutboundProxyCfg   *pOutboundCfg,
    IN RvInt32                          cfgStructSize);
```

PARAMETERS

hProxyTrx

The *ProxyTrx*.

pOutboundCfg

A pointer to the outbound proxy configuration structure with all the relevant details.

cfgStructSize

The size of the outbound proxy configuration structure.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxSetPersistency()

DESCRIPTION

Changes the *ProxyTrx* persistency definition at runtime. This function receives a Boolean value that indicates whether or not the application wishes this *ProxyTrx* to be persistent. A persistent *ProxyTrx* object will try to locate a suitable connection in the connection hash before opening a new connection.

SYNTAX

```
RvStatus RvProxyCoreTrxSetPersistency (
    IN RvProxyTrxHandle    hProxyTrx,
    IN RvBool              bIsPersistent);
```

PARAMETERS

hProxyTrx

The *ProxyTrx*.

bIsPersistent

Determines whether or not the *ProxyTrx* will try to use a persistent connection.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxSetViaBranch()

DESCRIPTION

Sets the top Via branch of the *ProxyTrx*. The *ProxyTrx* will add the branch to the top Via header of outgoing requests.

SYNTAX

```
RvStatus RvProxyCoreTrxSetViaBranch (
    IN RvProxyTrxHandle    hProxyTrx,
    IN RvChar*             strBranch,
    IN RPOOL_Ptr*          pRpoolViaBranch);
```

PARAMETERS

hProxyTrx

The *ProxyTrx* object.

strBranch

The Via branch to add to the top Via header. This parameter is ignored for response messages or when the *bAddTopVia* parameter in the RvProxyCoreTrxSendMsg() function is RV_FALSE.

pRpoolViaBranch

The branch supplied on a page. You should set this parameter to NULL if the branch was supplied as a string.

RETURN VALUES

Returns RvStatus.

RvProxyCoreTrxSetForceOutboundAddrFlag()

DESCRIPTION

Sets the ForceOutboundAddr flag. This flag forces the *ProxyTrx* to send the request to the outbound address regardless of the message content.

SYNTAX

```
RvStatus RvProxyCoreTrxSetForceOutboundAddrFlag(
    IN RvProxyTrxHandle     hProxyTrx,
    IN RvBool               bForceOutboundAddr);
```

PARAMETERS

hProxyTrx

The handle to the *ProxyTrx*.

bForceOutboundAddr

The flag value to set.

RETURN VALUES

Returns RvStatus.

REGISTER SERVER MODULE

The Register Server module acts as a registrar for handling Register requests. With the Register Server API functions, you can decide whether to accept or reject a Register request and manipulate the registration data. The Register Server module uses the Location Database module functions for handling local Register requests.

This part includes the following section:

- Register Server Functions

3

REGISTER SERVER FUNCTIONS

WHAT'S IN THIS SECTION

This section contains the Register Server functions, which are grouped as follows:

- Register Server Manager Functions
- Register Server Object Functions
- Register Server Data Object Functions

Register Server Manager Functions

The Register Server Manager (*RegServerMgr*) is responsible for managing the Register Server Objects (*RegServerObj*). The Register Server Manager API functions enable you to set the event handler for the *RegServerObj*. These functions are included in the *RvProxyRegServerMgr.h* header file.

The Register Server Manager API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control function is:

- RvProxyRegServerMgrAttachAppObj()

RvProxyRegServerMgrAttachAppObj()

DESCRIPTION

Attaches the application owner to the *RegServerMgr*.

SYNTAX

```
RvStatus RvProxyRegServerMgrAttachAppObj(
    IN RvProxyRegServerMgrHandle       hRegServerMgr,
    IN RvProxyRegServerMgrAppHandle    hAppRegServerMgr);
```

PARAMETERS

hRegServerMgr

The handle to the *RegServerMgr*.

hAppRegServerMgr

The application object handle to attach as the *RegServerMgr* owner.

RETURN VALUES

Returns RvStatus.

Get and Set Functions

The Get and Set functions are:

- RvProxyRegServerMgrSetEvHandler()
- RvProxyRegServerMgrGetResourceStatus()
- RvProxyRegServerMgrGetAppObj()
- RvProxyRegServerMgrSetLocationDBMgrHandle()
- RvProxyRegServerMgrGetLocationDBMgrHandle()
- RvProxyRegServerMgrLocationDBSetInterface()
- RvProxyRegServerMgrLocationDBGetInterface()
- RvProxyRegServerMgrSetPASCMgrHandle()
- RvProxyRegServerMgrGetPASCSCMgrHandle()
- RvProxyRegServerMgrPASCSetInterface()
- RvProxyRegServerMgrPASCGetInterface()
- RvProxyRegServerMgrGetServerInstance()

RvProxyRegServerMgrSetEvHandler()

DESCRIPTION

Sets the *RegServerMgr* callback functions.

SYNTAX

```
RvStatus RvProxyRegServerMgrSetEvHandler(
    IN RvProxyRegServerMgrHandle     hMgr,
    IN RvProxyRegServerEvHandler*    pHandler,
    IN RvUint32                      size);
```

PARAMETERS

hMgr

The handle of the *RegServerMgr* to which to set the event handler.

pHandler

The pointer to the Register Server event handler structure.

size

The size of the RvProxyRegServerEvHandler structure.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerMgrGetResourceStatus()

DESCRIPTION

Returns the status of the *RegServerMgr* resources used.

Note This function is for Debug compilation mode only.

SYNTAX

```
RvStatus RvProxyRegServerMgrGetResourceStatus(
    IN  RvProxyRegServerMgrHandle    hRegServerMgr,
    OUT RvProxyRegServerResource*    pResourcesStatus);
```

PARAMETERS

hRegServerMgr

The handle to the *RegServerMgr*.

pResourcesStatus

The structure of the resources.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerMgrGetAppObj()

DESCRIPTION

Gets the handle of the *RegServerMgr* application handle.

SYNTAX

```
RvStatus RvProxyRegServerMgrGetAppObj(
    IN  RvProxyRegServerMgrHandle        hRegServerMgr,
    OUT RvProxyRegServerMgrAppHandle*    phAppRegServerMgr);
```

PARAMETERS

hRegServerMgr

The *RegServerMgr* handle.

phAppRegServerMgr

The handle of the application owner of the *RegServerMgr*.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerMgrSetLocationDBMgrHandle()

DESCRIPTION

Sets a *LocationDBMgr* handle.

SYNTAX

```
RvStatus RvProxyRegServerMgrSetLocationDBMgrHandle(
    IN RvProxyRegServerMgrHandle     hRegServerMgr,
    IN RvProxyLocationDBMgrHandle    hLocationDbMgr);
```

PARAMETERS

hRegServerMgr

The *RegServerMgr* handle.

hLocationDbMgr

The *LocationDBMgr* handle. Use NULL to remove it.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerMgrGetLocationDBMgrHandle()

DESCRIPTION

Returns the handle to the *LocationDBMgr*, if it exists.

SYNTAX

```
RvStatus RvProxyRegServerMgrGetLocationDBMgrHandle(
    IN  RvProxyRegServerMgrHandle      hRegServerMgr,
    OUT RvProxyLocationDBMgrHandle*    hLocationDbMgr);
```

PARAMETERS

hRegServerMgr

The *RegServerMgr* handle.

hLocationDbMgr

The handle to the *LocationDBMgr*.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerMgrLocationDBSetInterface()

DESCRIPTION

Sets the *LocationDB* handling functions. The *RegServer* uses these functions to access the *LocationDB* implementation.

SYNTAX

```
RvStatus RvProxyRegServerMgrLocationDBSetInterface(
   IN RvProxyRegServerMgrHandle      hRegServerMgr,
   IN RvProxyLocationDBInterface*    pLocationDBFuncHandler,
   IN RvUint32                       size);
```

PARAMETERS

hRegServerMgr

The *RegServerMgr* handle.

pLocationDBFuncHandler

The handle to a structure that contains the pointers to the functions.

size

The size of the structure that holds the pointers to the functions.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerMgrLocationDBGetInterface()

DESCRIPTION

Returns the handle to the locationDBFunc structure that holds the implementations of the *LocationDB* interface functions.

SYNTAX

```
RvStatus RvProxyRegServerMgrLocationDBGetInterface (
    IN  RvProxyRegServerMgrHandle       hRegServerMgr,
    OUT RvProxyLocationDBInterface**    pLocationDBFunc);
```

PARAMETERS

hRegServerMgr

The *RegServerMgr* handle.

pLocationDBFunc

The handle to the locationDBFunc structure.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerMgrGetPASCSCMgrHandle()

DESCRIPTION

Returns the handle to the *PASCMgr*, if it exists.

SYNTAX

```
RvStatus RvProxyRegServerMgrGetPASCMgrHandle(
    IN  RvProxyRegServerMgrHandle    hRegServerMgr,
    OUT void**                       hPASCMgr);
```

PARAMETERS

hRegServerMgr

The *RegServerMgr* handle.

hPASCMgr

The *PASCMgr* handle.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerMgrSetPASCMgrHandle()

DESCRIPTION

Sets a handle to the *PASCMgr*.

SYNTAX

```
RvStatus RvProxyRegServerMgrSetPASCMgrHandle(
    IN RvProxyRegServerMgrHandle    hRegServerMgr,
    IN void**                       hPASCMgr);
```

PARAMETERS

hRegServerMgr

The *RegServerMgr* handle.

hPASCMgr

The *PASCMgr* handle.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerMgrPASCSetInterface()

DESCRIPTION

Sets the Presence Agent Component handling functions. The *RegServer* uses these functions to access the Presence Agent Component implementations.

SYNTAX

```
RvStatus RvProxyRegServerMgrPASCSetInterface(
    IN RvProxyRegServerMgrHandle       hRegServerMgr,
    IN RvProxyRegServerPASCInterface*  pPASCFuncHandler,
    IN RvUint32                        size);
```

PARAMETERS

hRegServerMgr

The *RegServerMgr* handle.

pPASCFuncHandler

The handle to the structure that contains the pointers to these functions.

size

The size of the structure that holds the pointers to the functions.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerMgrPASCGetInterface()

DESCRIPTION

Returns the structure that holds pointers to the interface function implementation of the Presence Agent Component.

SYNTAX

```
RvStatus RvProxyRegServerMgrPASCGetInterface (
    IN  RvProxyRegServerMgrHandle          hRegServerMgr,
    OUT RvProxyRegServerPASCInterface**    pPASCFunc);
```

PARAMETERS

hRegServerMgr

The *RegServerMgr* handle.

pPASCFunc

A pointer to the structure of the Presence Agent functions.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerMgrGetServerInstance()

DESCRIPTION

Gets the handle of the *SIPServerMgr*.

SYNTAX

```
RvStatus RvProxyRegServerMgrGetServerInstance(
    IN  RvProxyRegServerMgrHandle    hRegServerMgr,
    OUT void**                       phSipServerMgr);
```

PARAMETERS

hRegServerMgr

The *RegServerMgr* handle.

phSipServerMgr

The handle of the *SIPServerMgr*.

RETURN VALUES

Returns RvStatus.

Register Server Object Functions

The Register Server Object API functions enable you to accept or reject Register requests according to the application policy, and to access the Register Server Object parameters. These functions are included in the *RvProxyRegServerObj.h* header file.

The Register Server Object API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvProxyRegServerObjAcceptRegistration()
- RvProxyRegServerObjRejectRegistration()
- RvProxyRegServerObjTerminate()
- RvProxyRegServerObjDisregardFailure()

RvProxyRegServerObjAcceptRegistration()

DESCRIPTION

Accepts the Registration request and processes and handles the registration according to the SIP RFC rules. If the *bUpdateLocationDB* parameter is TRUE, the record is updated in the Location Database according to the Registration request. If the update is successful, a response will be sent with the given response code and reason. If the given response code is 0, a default response code is sent (200). The 200 response will include all user registrations found in the Location Database as contact headers. If there was a failure in updating the Location Database Component, a 500 response is sent.

This function can be called from the VALID_REQUEST state.

SYNTAX

```
RvStatus RvProxyRegServerObjAcceptRegistration(
    IN RvProxyRegServerObjHandle    hRegServerObj,
    IN RvUint16                     responseCode,
    IN RvChar*                      strReason,
    IN RvBool                       bUpdateLocationDB);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj* responsible for this registration.

responseCode

The response code that should be sent with the acceptance. If the response code is 0, a default value is used.

strReason

The reason for accepting the registration. This parameter is a null terminating string.

bUpdateLocationDB

Specifies whether or not the Location Database should be updated with the registration.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerObjRejectRegistration()

DESCRIPTION

Rejects the Registration request. A response is sent with the response code and the reason is supplied. If the response code is 0, a default value of 400 is used. In the case of non-authentication, 401/407 is used. This function can be called from the VALID_REQUEST or the INVALID_REQUEST states.

SYNTAX

```
RvStatus RvProxyRegServerObjRejectRegistration(
    IN RvProxyRegServerObjHandle    hRegServerObj,
    IN RvUint16                     responseCode,
    IN RvChar*                      strReason);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj* responsible for this registration.

responseCode

The response code that should be sent with the rejection. If the value is 0, a default value is used.

strReason

The reason for the rejection as a null terminated string.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerObjTerminate()

DESCRIPTION

Terminates the *RegServerObj*.

SYNTAX

```
RvStatus RvProxyRegServerObjTerminate(
    IN RvProxyRegServerObjHandle    hRegServerObj);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj* to be terminated.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerObjDisregardFailure()

DESCRIPTION

Disregards an authentication or validation failure, and moves to the VALID_REQ state from the INVALID state. This function can be called from the INVALID state only.

SYNTAX

```
RvStatus RvProxyRegServerObjDisregardFailure(
    IN RvProxyRegServerObjHandle    hRegServerObj);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj*.

RETURN VALUES

Returns RvStatus.

Get and Set Functions

The Get and Set functions are:

- RvProxyRegServerObjGetState()
- RvProxyRegServerObjGetRecord()
- RvProxyRegServerObjGetRegServerMgr()
- RvProxyRegServerObjGetRcvdMsg()
- RvProxyRegServerObjGetbAuthenticate()
- RvProxyRegServerObjSetbAuthenticate()
- RvProxyRegServerObjGetAppHandle()
- RvProxyRegServerObjSetAppHandle()
- RvProxyRegServerObjAddAuthHeaderToResponse()
- RvProxyRegServerObjGetReceivedFromAddress()
- RvProxyRegServerObjGetReceivedLocalAddress()

RvProxyRegServerObjGetState()

DESCRIPTION

Returns the current state of the *RegServerObj*.

SYNTAX

```
RvProxyRegServerObjState RvProxyRegServerObjGetState(
    RvProxyRegServerObjHandle    hRegServerObj);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj*.

RETURN VALUES

Returns the *RegServerObj* state or RVPROXY_REGSERVER_OBJ_STATE_UNDEFINED for error.

RvProxyRegServerObjGetRecord()

DESCRIPTION

Returns the record in the *RegServerObj* that holds the registration information.

SYNTAX

```
RvProxyRegServerRecordHandle RvProxyRegServerObjGetRecord(
    RvProxyRegServerObjHandle    hRegServerObj);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj*.

RETURN VALUES

Returns the *RegServerRecord* or NULL for error.

RvProxyRegServerObjGetRegServerMgr()

DESCRIPTION

Returns the handle of the *RegServerMgr*.

SYNTAX

```
RvStatus RvProxyRegServerObjGetRegServerMgr(
    IN  RvProxyRegServerObjHandle     hRegServerObj,
    OUT RvProxyRegServerMgrHandle*    phRegServerMgr);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj*.

phRegServerMgr

The handle to the *RegServerMgr*.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerObjGetRcvdMsg()

DESCRIPTION

Returns the handle of the received message. The message can be retrieved only until after the *RegServerObj* reaches the VALID or INVALID state.

SYNTAX

```
RvStatus RvProxyRegServerObjGetRcvdMsg(
    IN  RvProxyRegServerObjHandle    hRegServerObj,
    OUT RvSipMsgHandle*              phRcvdMsg);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj*.

phRcvdMsg

The message that was received and handled by this *RegServerObj*.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerObjGetbAuthenticate()

DESCRIPTION

Returns whether or not the Register request should be authenticated.

SYNTAX

```
RvStatus RvProxyRegServerObjGetbAuthenticate(
    IN  RvProxyRegServerObjHandle    hRegServerObj,
    OUT RvBool*                      bAuthenticate);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj*.

bAuthenticate

Indicates whether or not authentication is required for this *RegServerObj*.

RETURN VALUES

Returns the Boolean value that indicates whether or not to authenticate the Register request.

RvProxyRegServerObjSetbAuthenticate()

DESCRIPTION

Indicates whether or not the Register request should be authenticated.

SYNTAX

```
RvStatus RvProxyRegServerObjSetbAuthenticate(
    IN RvProxyRegServerObjHandle    hRegServerObj,
    IN RvBool                       bAuthenticate);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj*.

bAuthenticate

The value to set in bAuthenticate.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerObjGetAppHandle()

DESCRIPTION

Returns the application handle of the *RegServerObj*.

SYNTAX

```
RvStatus RvProxyRegServerObjGetAppHandle(
    IN  RvProxyRegServerObjHandle        hRegServerObj,
    OUT RvProxyRegServerObjAppHandle*    hAppRegServerObj);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj*.

hAppRegServerObj

The handle to the application owner of the *RegServerObj*.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerObjSetAppHandle()

DESCRIPTION

Sets the application handle of the *RegServerObj*.

SYNTAX

```
RvStatus RvProxyRegServerObjSetAppHandle(
    IN RvProxyRegServerObjHandle       hRegServerObj,
    IN RvProxyRegServerObjAppHandle    hAppRegServerObj);
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj*.

hAppRegServerObj

The handle to the application owner of the *RegServerObj*.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerObjAddAuthHeaderToResponse()

DESCRIPTION

Indicates whether or not the specified *RegServerObj* should add a WWW-Authentication header or Proxy-Authentication header to to an initiated 401/407 (respectively) response message.

SYNTAX

```
RvStatus RvProxyRegServerObjAddAuthHeaderToResponse(
    IN  RvProxyRegServerObjHandle    hRegServerObj,
    IN  RV_BOOL                      bAddAuthHeader)
```

PARAMETERS

hRegServerObj

The handle to the *RegServerObj* to set this attribute.

bAddAuthHeader

If RV_TRUE, adds the Un-Authenticated header to the 401/407 response message. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerObjGetReceivedFromAddress()

DESCRIPTION

Gets the address from which the last message was received.

SYNTAX

```
RvStatus RvProxyRegServerObjGetReceivedFromAddress(
    IN  RvProxyRegServerObjHandle    hRegServerObj,
    OUT RvSipTransportAddr*          pAddr)
```

PARAMETERS

hRegServerObj

The handle to *RegServerObj* to set this attribute.

pAddr

The basic details about the received From address.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerObjGetReceivedLocalAddress()

DESCRIPTION

Gets the local address on which the request was received.

SYNTAX

```
RvProxyRegServerObjGetReceivedLocalAddress(
    IN  RvProxyRegServerObjHandle    hRegServerObj,
    OUT RvSipTransport*              peTransportType,
    OUT RvSipTransportAddressType*   peAddressType,
    OUT RvChar*                      pszLocalAddress,
    OUT RvUint16*                    pLocalPort)
```

PARAMETERS

pRegServerObj

The andle to the *RegServerObj* to return its received local address.

peTransportType

The transport type (UDP, TCP).

peAddressType

The address type (IP4 or IP6).

pszLocalAddress

The local address.

pLocalPort

The local port.

RETURN VALUES

Returns RvStatus.

Register Server Data Object Functions

The Register Server Data Objects API functions enable you to create and control data structures that the Register Server uses for holding registration information. The API functions allow you to construct and destruct these data structures and change or get data from them. These functions are included in the *RvProxyRegServerDataObjects.h* header file.

The data structures are:

- **RegServerRecord**—holds register information for a user as a list of bindings and identified by a key. See Register Server Record Functions.
- **RegServerBinding**—holds information of a single registration represented as a contact header in the registration request. See Register Server Binding Functions.
- **RegServerKey**—the key that identifies the record. See Register Server Key Functions.

REGISTER SERVER RECORD FUNCTIONS

The Register Server Record API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvProxyRegServerRecordConstruct()
- RvProxyRegServerRecordDestruct()
- RvProxyRegServerRecordRemoveBinding()
- RvProxyRegServerRecordManipulateExpiresBindings()

RvProxyRegServerRecordConstruct()

DESCRIPTION

Constructs a *RegServerRecord*. The record is allocated in the array of records. The binding list and *RegServerKey* of the *RegServerRecord* are also constructed.

SYNTAX

```
RvProxyRegServerRecordHandle
RvProxyRegServerRecordConstruct(
    IN RvProxyRegServerMgrHandle    hRegServerMgr,
    IN HRPOOL                       hPool);
```

PARAMETERS

hRegServerMgr

The handle to the *RegServerMgr*.

hPool

The application should supply a pool of pages. A page from this pool will be used for constructing the *RegServerRecord*. If this parameter is NULL, the internal memory pool of the Register Server Module constructs the record.

RETURN VALUES

Returns the *RegServerRecord* or NULL for error.

RvProxyRegServerRecordDestruct()

DESCRIPTION

Destructs a *RegServerRecord*. The key and binding list of the record is destructed, and the *RegServerRecord* is de-allocated from the array of records. The page with which the record was constructed will be freed.

SYNTAX

```
RvStatus RvProxyRegServerRecordDestruct(
    IN RvProxyRegServerRecordHandle    hRegServerRecord,
    IN HRPool                          hPool);
```

PARAMETERS

hRegServerRecord

The *RegServerRecord* to destruct.

hPool

The pool of memory pages with which the *RegServerRecord* was constructed. If the *RegServerRecord* was constructed with the internal memory pool of the Register Server Module, this parameter should be NULL.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerRecordRemoveBinding()

DESCRIPTION

Removes a *RegServerBinding* from a *RegServerRecord*. The *RegServerBinding* is no longer valid after it was removed it since it was de-allocated.

SYNTAX

```
RvStatus RvProxyRegServerRecordRemoveBinding(
    IN RvProxyRegServerRecordHandle     hRegServerRecord,
    IN RvProxyRegServerBindingHandle    hBinding);
```

PARAMETERS

hRegServerRecord

The handle to the *RegServerRecord* from which the *RegServerBinding* should be removed.

hBinding

The handle to the *RegServerBinding* to be removed.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerRecordManipulateExpiresBindings()

DESCRIPTION

Runs over the bindings in *hRegServerRec* and removes or leaves the expires bindings according to the *bRemoveExpires* value.

SYNTAX

```
RvStatus RvProxyRegServerRecordManipulateExpiresBindings(
    IN  RvProxyRegServerRecordHandle    hRegServerRec,
    IN  RvBool                          bRemoveExpires)
```

PARAMETERS

hRegServerRec

The bindings to handle.

bRemoveExpires

If RV_TRUE, remove the expires bindings. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

The Get and Set functions are:

- RvProxyRegServerRecordGetKey()
- RvProxyRegServerRecordSetKey()
- RvProxyRegServerRecordGetHeadBinding()
- RvProxyRegServerRecordGetTailBinding()
- RvProxyRegServerRecordGetNextBinding()
- RvProxyRegServerRecordGetPrevBinding()

RvProxyRegServerRecordGetKey()

DESCRIPTION

Returns the *RegServerKey* handle in the *RegServerRecord*.

SYNTAX

```
RvProxyRegServerKeyHandle RvProxyRegServerRecordGetKey(
    IN RvProxyRegServerRecordHandle    hRegServerRecord);
```

PARAMETERS

hRegServerRecord

The *RegServerRecord* from which to get the *RegServerKey*.

RETURN VALUES

Returns the *RegServerKey* handle or NULL for error.

RvProxyRegServerRecordSetKey()

DESCRIPTION

Sets a *RegServerKey* in the *RegServerRecord*. If a *RegServerKey* already exists, the data from the *RegServerKey* argument is copied into the *RegServerKey* of the *RegServerRecord*, on the page of the record. If no *RegServerKey* exists, a new one is allocated and the data is copied.

SYNTAX

```
RvStatus RvProxyRegServerRecordSetKey(
    IN RvProxyRegServerRecordHandle    hRegServerRecord,
    IN RvProxyRegServerKeyHandle       hKey);
```

PARAMETERS

hRegServerRecord

The *RegServerRecord* to be set.

hKey

The *RegServerKey* from which to copy the values.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerRecordGetHeadBinding()

DESCRIPTION

Returns the head *RegServerBinding* from the binding list of the *RegServerRecord*.

SYNTAX

```
RvStatus RvProxyRegServerRecordGetHeadBinding(
    IN  RvProxyRegServerRecordHandle      hRegServerRecord,
    OUT RvProxyRegServerBindingHandle*    phHeadBinding);
```

PARAMETERS

hRegServerRecord

The *RegServerRecord* with the binding list.

phHeadBinding

The head *RegServerBinding* of the list. If the list is empty, points to NULL.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerRecordGetTailBinding()

DESCRIPTION

Returns the last *RegServerBinding* from the binding list of the *RegServerRecord*.

SYNTAX

```
RvStatus RvProxyRegServerRecordGetTailBinding(
    IN  RvProxyRegServerRecordHandle      hRegServerRecord,
    OUT RvProxyRegServerBindingHandle*    phTailBinding);
```

PARAMETERS

hRegServerRecord

The *RegServerRecord* with binding list.

phTailBinding

The last *RegServerBinding* of the list. If the list is empty, points to NULL.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerRecordGetNextBinding()

DESCRIPTION

Returns the next *RegServerBinding* in the binding list of the *RegServerRecord*.

SYNTAX

```
RvStatus RvProxyRegServerRecordGetNextBinding(
    IN  RvProxyRegServerRecordHandle     hRegServerRecord,
    IN  RvProxyRegServerBindingHandle    hCurrBinding,
    OUT RvProxyRegServerBindingHandle*   phNextBinding);
```

PARAMETERS

hRegServerRecord

The *RegServerRecord* with binding list.

hCurrBinding

The current *RegServerBinding* in the list of which the next binding should be returned.

phNextBinding

The next *RegServerBinding* in the list. If the *hCurrBinding* is the last binding in the list, points to NULL.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerRecordGetPrevBinding()

DESCRIPTION

Returns the previous *RegServerBinding* from the binding list of the *RegServerRecord*.

SYNTAX

```
RvStatus RvProxyRegServerRecordGetPrevBinding(
    IN  RvProxyRegServerRecordHandle      hRegServerRecord,
    IN  RvProxyRegServerBindingHandle     hCurrBinding,
    OUT RvProxyRegServerBindingHandle*    phPrevBinding);
```

RETURN VALUES

PARAMETERS

hRegServerRecord

The *RegServerRecord* with binding list.

hCurrBinding

The current *RegServerBinding* in the list of which its previous element should be returned.

phPrevBinding

The previous *RegServerBinding* in list. If the *hCurrBinding* is the first binding in list, points to NULL.

Register Server Binding Functions

The Register Server Binding API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control function is:

- RvProxyRegServerBindingConstructInRecord()

RvProxyRegServerBindingConstructInRecord()

DESCRIPTION

Adds a new *RegServerBinding* to an existing *RegServerRecord*. The *RegServerBinding* is allocated and added to the binding list of the *RegServerRecord*. After constructing the *RegServerBinding*, the application should set its values using the Register Server Binding API Set functions. The Contact header, CSeq header and Call ID parameters must be set with values.

SYNTAX

```
RvStatus RvProxyRegServerBindingConstructInRecord(
    IN  RvProxyRegServerRecordHandle     hRegServerRecord,
    IN  RvProxyElemLocation              bindingLocation,
    IN  RvProxyRegServerBindingHandle    currBinding,
    OUT RvProxyRegServerBindingHandle*   phNewBinding);
```

PARAMETERS

hRegServerRecord

The *RegServerRecord* in which to construct the new *RegServerBinding*.

bindingLocation

The location of the new *RegServerBinding* in the binding list of the *RegServerRecord*.

currBinding

If the location is next/prev, the reference binding in the list.

phNewBinding

The new *RegServerBinding* that was constructed in the *RegServerRecord*.

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

The Get and Set functions are:

- RvProxyRegServerBindingGetContactHeader()
- RvProxyRegServerBindingSetContactHeader()
- RvProxyRegServerBindingGetCSeqHeader()
- RvProxyRegServerBindingSetCSeqHeader()
- RvProxyRegServerBindingGetCallIdStr()
- RvProxyRegServerBindingSetCallIdStr()

RvProxyRegServerBindingGetContactHeader()

DESCRIPTION

Returns the Contact header from the *RegServerBinding*.

SYNTAX

```
RvSipContactHeaderHandle
RvProxyRegServerBindingGetContactHeader(
    IN RvProxyRegServerBindingHandle    hRegServerBinding);
```

PARAMETERS

hRegServerBinding

The handle to the *RegServerBinding*.

RETURN VALUES

Returns the contact header. NULL for failure or if there is no contact header in this binding.

RvProxyRegServerBindingSetContactHeader()

DESCRIPTION

Sets the Contact header in the *RegServerBinding*. The new Contact header is copied into the *RegServerBinding*, on the memory page of the *RegServerRecord*.

SYNTAX

```
RvStatus RvProxyRegServerBindingSetContactHeader(
    IN RvProxyRegServerBindingHandle    hRegServerBinding,
    IN RvProxyRegServerRecordHandle     hRecord,
    IN RvSipContactHeaderHandle         hContactHeader);
```

PARAMETERS

hRegServerBinding

The handle to the *RegServerBinding*.

hRecord

The *RegServerRecord* with the *RegServerBinding* that should be set.

hContactHeader

The Contact header which should be copied into the binding.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerBindingGetCSeqHeader()

DESCRIPTION

Returns the CSeq header from the *RegServerBinding*.

SYNTAX

```
RvSipCSeqHeaderHandle RvProxyRegServerBindingGetCSeqHeader(
     IN RvProxyRegServerBindingHandle    hRegServerBinding);
```

PARAMETERS

hRegServerBinding

The handle to the *RegServerBinding*.

RETURN VALUES

Returns the CSeq header. NULL for failure, or if there is no CSeq header in this binding.

RvProxyRegServerBindingSetCSeqHeader()

DESCRIPTION

Sets the CSeq header in the *RegServerBinding*. The new CSeq header is copied into the *RegServerBinding* on the memory page of the *RegServerRecord*.

SYNTAX

```
RvStatus RvProxyRegServerBindingSetCSeqHeader(
    IN RvProxyRegServerBindingHandle    hRegServerBinding,
    IN RvProxyRegServerRecordHandle     hRecord,
    IN RvSipCSeqHeaderHandle            hCSeqHeader);
```

PARAMETERS

hRegServerBinding

The handle to the *RegServerBinding*.

hRecord

The *RegServerRecord* with the *RegServerBinding* that should be set.

hCSeqHeader

The CSeq header which should be copied into the *RegServerBinding*.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerBindingGetCallIdStr()

DESCRIPTION

Returns the call ID string from the *RegServerBinding*.

SYNTAX

```
RvStatus RvProxyRegServerBindingGetCallIdStr(
    IN    RvProxyRegServerBindingHandle   hRegServerBinding,
    INOUT RvUint32*                       callIdStrLen,
    OUT   RvChar*                         callIdStr);
```

PARAMETERS

hRegServerBinding

The handle to the *RegServerBinding*.

callIdStrLen

The length of the buffer that the application supplies. If the buffer length is smaller than the call ID length, RV_InsufficientBuffer error is returned. Otherwise, the actual length of the call ID string is returned.

callIdStr

The string where the call ID is returned.

RETURN VALUES

Returns the call ID string, or NULL for failure.

RvProxyRegServerBindingSetCallIdStr()

DESCRIPTION

Copies the call ID string into the *RegServerBinding*. The string is copied into the binding on the memory page of the *RegServerRecord*.

SYNTAX

```
RvStatus RvProxyRegServerBindingSetCallIdStr(
    IN RvProxyRegServerBindingHandle    hRegServerBinding,
    IN RvProxyRegServerRecordHandle     hRecord,
    IN RvChar*                          callIdStr);
```

PARAMETERS

hRegServerBinding

The handle to the *RegServerBinding*.

hRecord

The *RegServerRecord* with the *RegServerBinding* that should be set.

callIdStr

The call ID string to be set in the *RegServerBinding*.

RETURN VALUES

Returns RvStatus.

Register Server Key Functions

The Register Server Key API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvProxyRegServerKeyConstruct()
- RvProxyRegServerKeyDestruct()

RvProxyRegServerKeyConstruct()

DESCRIPTION

Constructs a new stand-alone *RegServerKey*. This key is not attached to a *RegServerRecord*, and the data of the *RegServerKey* is saved on the page given as an argument. This page should not be freed while using this key. In order to destruct this *RegServerKey*, call RvProxyRegServerKeyDestruct() and then free the page.

SYNTAX

```
RvProxyRegServerKeyHandle RvProxyRegServerKeyConstruct(
    IN RvProxyRegServerMgrHandle    hRegServerMgr,
    IN HRPOOL                       hPool,
    IN HPAGE                        hPage);
```

PARAMETERS

hRegServerMgr

The handle to the *RegServerMgr*.

hPool

The memory pool of the page.

hPage

The page where the TO header is written.

RETURN VALUES

Returns the new *RegServerKey* handle, or NULL for failure.

RvProxyRegServerKeyDestruct()

DESCRIPTION

Destructs *RegServerKey* and de-allocates it from the array of the *RegServerKey* objects. After destructing the *RegServerKey*, the memory page used to create this *RegServerKey* should be freed by the application which supplied this page for constructing the *RegServerKey*.

SYNTAX

```
RvStatus RvProxyRegServerKeyDestruct(
    IN RvProxyRegServerKeyHandle    hKey,
    IN RvProxyRegServerMgrHandle    hRegServerMgr);
```

PARAMETERS

hKey

The *RegServerKey* handle to destruct.

hRegServerMgr

The handle to the *RegServerMgr*.

RETURN VALUES

Returns RvStatus.

Get and Set Functions

The Get and Set functions are:

- RvProxyRegServerKeyGetToHeader()
- RvProxyRegServerKeySetToHeader()

RvProxyRegServerKeyGetToHeader()

DESCRIPTION

Returns the TO header from the *RegServerKey*.

SYNTAX

```
RvSipPartyHeaderHandle RvProxyRegServerKeyGetToHeader(
    IN RvProxyRegServerKeyHandle    hRegServerKey);
```

PARAMETERS

hRegServerKey

The handle to the *RegServerKey*.

RETURN VALUES

Returns the handle of the TO header. NULL for failure or if there is no TO header that was set to this Key.

RvProxyRegServerKeySetToHeader()

DESCRIPTION

Sets the TO header in the *RegServerKey*. The TO header will be copied to the memory page of the *RegServerKey*.

SYNTAX

```
RvStatus RvProxyRegServerKeySetToHeader(
    IN RvProxyRegServerKeyHandle    hRegServerKey,
    IN RvSipPartyHeaderHandle       hToHeader);
```

PARAMETERS

hRegServerKey

The handle to the *RegServerKey*.

hToHeader

The new TO header to set to the *RegServerKey*.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerKeySetToHeader()

Server Authentication Module

The Sever Authentication module supplies authentication services for the various modules of the SIP Server. The Server Authentication Module interacts with the Security Component, and its API functions control the authentication process.

This part includes the following sections:

- Server Authentication Functions

4

SERVER AUTHENTICATION FUNCTIONS

WHAT'S IN THIS SECTION

This section contains the Server Authentication functions, which are grouped as follows:

- Server Authentication Manager Functions
- Server Authentication Functions

SERVER AUTHENTICATION MANAGER FUNCTIONS

The Server Authentication Manager (*ServerAuthMgr*) manages the collection of all Server Authentication objects (*ServerAuth*). The Server Authentication Manager API functions enable you to set the event handler functions for the authentication process. These functions are found in the *RvSipServerAuthMgr.h* header file.

The Server Authentication Manager API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control function is:

- RvSipServerAuthMgrAttachAppMgr()

RvSipServerAuthMgrAttachAppMgr()

DESCRIPTION

Sets the handle to the application owner of the *ServerAuthMgr*.

SYNTAX

```
RvStatus RvSipServerAuthMgrAttachAppMgr(
    IN RvSipServerAuthMgrHandle       hServerAuthMgr,
    IN RvSipServerAuthMgrAppHandle    hAppAuthMgr);
```

PARAMETERS

hServerAuthMgr

The handle to the *ServerAuthMgr*.

hAppAuthMgr

The application owner to set.

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

The Get and Set functions are:

- RvSipServerAuthMgrGetAppMgr()
- RvSipServerAuthMgrSetSecurityMgrHandle()
- RvSipServerAuthMgrGetSecurityMgrHandle()
- RvSipServerAuthMgrSecuritySetInterface()
- RvSipServerAuthMgrSecurityGetInterface()
- RvSipServerAuthMgrGetResourceStatus()
- RvSipServerAuthMgrGetServerInstance()

RvSipServerAuthMgrGetAppMgr()

DESCRIPTION

Returns the handle to the application owner of the *ServerAuthMgr*.

SYNTAX

```
RvStatus RvSipServerAuthMgrGetAppMgr(
    IN  RvSipServerAuthMgrHandle        hServerAuthMgr,
    OUT RvSipServerAuthMgrAppHandle*    hAppAuthMgr);
```

PARAMETERS

hServerAuthMgr

The handle to the *ServerAuthMgr*.

hAppAuthMgr

The returned application owner.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthMgrSetSecurityMgrHandle()

DESCRIPTION

Sets a Security Component Manager (*SecurityMgr*) handle to the *ServerAuthMgr*.

SYNTAX

```
RvStatus RvSipServerAuthMgrSetSecurityMgrHandle(
    IN RvSipServerAuthMgrHandle        hServerAuthMgr,
    IN RvSipServerSecurityMgrHandle    hSecurityMgr);
```

PARAMETERS

hServerAuthMgr

The handle to the *ServerAuthMgr*.

hSecurityMgr

The *SecurityMgr* handle. To remove it, use NULL.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthMgrGetSecurityMgrHandle()

DESCRIPTION

Returns the handle to the *SecurityMgr*, if it exists.

SYNTAX

```
RvStatus RvSipServerAuthMgrGetSecurityMgrHandle(
    IN  RvSipServerAuthMgrHandle         hServerAuthMgr,
    OUT RvSipServerSecurityMgrHandle*    hSecurity);
```

PARAMETERS

hServerAuthMgr

The handle to the *ServerAuthMgr*.

hSecurity

The handle to the *SecurityMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthMgrSecuritySetInterface()

DESCRIPTION

Sets the Security Component interface functions. The Server Authentication module uses the Security Component interface functions for authenticating requests.

SYNTAX

```
RvStatus RvSipServerAuthMgrSecuritySetInterface(
    IN RvSipServerAuthMgrHandle        hServerAuthMgr,
    IN RvSipServerSecurityInterface*   pSecurityFuncHandler,
    IN RvUint32                        size);
```

PARAMETERS

hServerAuthMgr

The handle of the *ServerAuthMgr*.

hSecurityFuncHandler

The handle to a structure that contains the pointers to the functions.

size

The size of the structure that holds the pointers to the functions.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthMgrSecurityGetInterface()

DESCRIPTION

Returns the handle to the Security Component interface structure.

SYNTAX

```
RvStatus RvSipServerAuthMgrSecurityGetInterface(
    IN  RvSipServerAuthMgrHandle          hServerAuthMgr,
    OUT RvSipServerSecurityInterface**    hSecurityFunc);
```

PARAMETERS

hServerAuthMgr

The handle to the *ServerAuthMgr*.

hSecurityFunc

The handle to the Security Component structure of the functions.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthMgrGetResourceStatus()

DESCRIPTION

Return the status of the *ServerAuthMgr* resources.

Note This function is for Debug compilation mode only.

SYNTAX

```
RvStatus RvSipServerAuthMgrGetResourceStatus(
    IN  RvSipServerAuthMgrHandle     hServerAuthMgr,
    OUT RvSipServerAuthResources*    pResourcesStatus);
```

PARAMETERS

hServerAuthMgr

The handle to *ServerAuthMgr*.

pResourcesStatus

The resource structure.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthMgrGetServerInstance()

DESCRIPTION

Gets the handle of the *ServerAuthMgr*.

SYNTAX

```
RvStatus RvSipServerAuthMgrGetServerInstance(
    IN  RvSipServerAuthMgrHandle    hServerAuthMgr,
    OUT void**                      phSipServerMgr);
```

PARAMETERS

hServerAuthMgr

ServerAuthMgr handle.

phSipServerMgr

The handle of the *ServerAuthMgr*.

RETURN VALUES

Returns RvStatus.

SERVER AUTHENTICATION FUNCTIONS

The Server Authentication object (*ServerAuth*) manages the authentication process for an incoming request. The *ServerAuth* uses the Security Component for the authentication process. The Server Authentication API functions allow you to control the progress of authentication and set and retrieve the parameters of the *ServerAuth*. These functions are found in the *RvSipServerAuthObj.h* header file.

The Server Authentication API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvSipServerAuthObjSetAuthResult()
- RvSipServerAuthObjSetPwd()

RvSipServerAuthObjSetAuthResult()

DESCRIPTION

Sets the authentication process result. The Security Component should call this function after it has an answer regarding the Authorization header that is in the *ServerAuth*. The *ServerAuth* will proceed according to the given result.

SYNTAX

```
RvStatus RvSipServerAuthObjSetAuthResult(
    IN RvSipServerAuthObjHandle         hServerAuthObj,
    IN RvSipServerSecurityAuthResult    eResult);
```

PARAMETERS

hServerAuthObj

The *ServerAuth* that is waiting for the authentication result.

eResult

The authentication result.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthObjSetPwd()

DESCRIPTION

Supplies a password in order to continue with the server-side authentication process. This function is valid only when the *ServerAuth* is in the AWAITING_PASSWORD state.

SYNTAX

```
RvStatus RvSipServerAuthObjSetPwd(
    IN RvSipServerAuthObjHandle    hServerAuthObj,
    IN const RvChar*               strPwd,
    IN RvBool                      bIsFound);
```

PARAMETERS

hServerAuthObj

The handle to the *ServerAuth* waiting for password.

strPwd

The supplied password.

bIsFound

An indication of whether or not the user in the authorization header was found in the Security Component database.

RETURN VALUES

Returns RvStatus.

Get and Set Functions

The Get and Set functions are:

- RvSipServerAuthObjGetAuthMgr()
- RvSipServerAuthObjGetSecurityMgr()
- RvSipServerAuthObjGetRequestMsg()
- RvSipServerAuthObjGetState()
- RvSipServerAuthObjGetAuthHeader()
- RvSipServerAuthObjSetAuthenticationHeader()
- RvSipServerAuthObjSetAuthResult()

RvSipServerAuthObjGetAuthMgr()

DESCRIPTION

Returns a handle to the *ServerAuthMgr*.

SYNTAX

```
RvStatus RvSipServerAuthObjGetAuthMgr(
    IN  RvSipServerAuthObjHandle    hServerAuthObj,
    OUT RvSipServerAuthMgrHandle    *phAuthMgr);
```

PARAMETERS

hServerAuthObj

The *ServerAuth*.

phAuthMgr

The returned *ServerAuthMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthObjGetSecurityMgr()

DESCRIPTION

Returns a handle to the *SecurityMgr*.

SYNTAX

```
RvStatus RvSipServerAuthObjGetSecurityMgr(
    IN  RvSipServerAuthObjHandle        hServerAuthObj,
    OUT RvSipServerSecurityMgrHandle    *phSecurityMgr);
```

PARAMETERS

hServerAuthObj

The *ServerAuth*.

phSecurityMgr

The returned *SecurityMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthObjGetRequestMsg()

DESCRIPTION

Returns the request message that is being authenticated.

SYNTAX

```
RvStatus RvSipServerAuthObjGetRequestMsg(
    IN  RvSipServerAuthObjHandle    hServerAuthObj,
    OUT RvSipMsgHandle              *phMsg);
```

PARAMETERS

hServerAuthObj

The *ServerAuth* responsible for authenticating the request message.

phMsg

The returned request message.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthObjGetState()

DESCRIPTION

Returns the current state of the *ServerAuth*. The state of the *ServerAuth* represents the state of the authentication process.

SYNTAX

```
RvSipServerAuthState RvSipServerAuthObjGetState(
    IN RvSipServerAuthObjHandle    hServerAuthObj);
```

PARAMETERS

hServerAuthObj

The *ServerAuth* whose state should be returned.

RETURN VALUES

The state of the *ServerAuth*.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthObjGetAuthHeader()

DESCRIPTION

Returns a handle to the Authorization header that the *ServerAuth* is currently checking.

SYNTAX

```
RvStatus RvSipServerAuthObjGetAuthHeader(
    IN  RvSipServerAuthObjHandle          hServerAuthObj,
    OUT RvSipAuthorizationHeaderHandle    *phAuth);
```

PARAMETERS

hServerAuthObj

The *ServerAuth* handle.

phAuth

The Authorization header that the *ServerAuth* is checking.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthObjSetAuthenticationHeader()

DESCRIPTION

Sets an Authentication header to be sent with the 401/407 response message in case the authentication fails. This function is valid only in the AWAITING_AUTH_HEADER state.

SYNTAX

```
RvStatus RvSipServerAuthObjSetAuthenticationHeader(
    IN RvSipServerAuthObjHandle    hServerAuthObj,
    IN void                        *pHeader);
```

PARAMETERS

hServerAuthObj

The *ServerAuth*.

pHeader

The Authentication header to be added to the 401/407 response message.

RETURN VALUES

Returns RvStatus.

RvSipServerAuthObjSetAuthResult()

DESCRIPTION

Reports about the authentication result. The Security Component should call this function after it has an answer regarding the Authorization header which is in the *ServerAuth*. The *ServerAuth* will proceed according to the given result.

SYNTAX

```
RvStatus RvSipServerAuthObjSetAuthResult(
    IN  RvSipServerAuthObjHandle          hServerAuthObj,
    IN  RvSipServerSecurityAuthResult     eResult)
```

PARAMETERS

hServerAuthObj

The *ServerAuth*waiting for the authentication result.

eResult

The authentication result.

RETURN VALUES

Returns RvStatus.

TRANSPORT MODULE

The Transport module allows you to control the Name Resolution process.

This part includes the following section:

- Transport Functions

5

TRANSPORT FUNCTIONS

WHAT'S IN THIS SECTION

The Transport layer of the RADVISION SIP Server Toolkit allows you to control the sending and receiving of messages over the network. This section contains Transport functions found in the *RvSipServerTransportMgr.h*, *RvSipTransport.h* and *RvSipTransportDNS.h* header files.

The functions included in this section are:

- SIP Server Transport Manager Functions
- SIP Stack Transport Functions
- DNS Transport Functions

SIP Server Transport Manager Functions

The SIP Server Transport Manager functions include:

- Get and Set Functions

Get and Set Functions

The Get and Set functions are:

- RvSipServerTransportMgrAttachAppMgr()
- RvSipServerTransportMgrGetAppMgrHandle()
- RvSipServerTransportMgrGetServerInstance()
- RvSipServerTransportMgrSetEvHandler()
- RvSipServerTransportMgrLIRAdd()
- RvSipServerTransportMgrLIRFindByAddress()
- RvSipServerTransportMgrLIRRemove()
- RvSipServerTransportMgrLIRGetFirst()
- RvSipServerTransportMgrLIRGetNext()
- RvSipServerTransportMgrGetNumOfLocalAddress()
- RvSipServerTransportMgrGetNumOfDomains()
- RvSipServerTransportMgrDomainAdd()
- RvSipServerTransportMgrDomainRemove()
- RvSipServerTransportMgrDomainGetFirst()
- RvSipServerTransportMgrDomainGetNext()

RvSipServerTransportMgrAttachAppMgr()

DESCRIPTION

Attaches an application manager handle to the Server Transport Manager (*ServerTransportMgr*).

SYNTAX

```
RV_Status RvSipServerTransportMgrAttachAppMgr(
  IN RvSipServerTransportMgrHandle      hServerTransportMgr,
  IN RvSipServerTransportMgrAppHandle   hAppMgr);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

hAppMgr

The application handle of the *ServerTransportMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrGetAppMgrHandle()

DESCRIPTION

Returns the application handle of the *ServerTransportMgr*.

SYNTAX

```
RV_Status RvSipServerTransportMgrGetAppMgrHandle(
  IN  RvSipServerTransportMgrHandle     hServerTransportMgr,
  OUT RvSipServerTransportMgrAppHandle* phAppMgr);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

phAppMgr

The application handle of the *ServerTransportMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrGetServerInstance()

DESCRIPTION

Gets the handle of the SIP Server Manager.

SYNTAX

```
RV_Status RvSipServerTransportMgrGetServerInstance(
   IN  RvSipServerTransportMgrHandle    hServerTransportMgr,
   OUT void**                           phSipServerMgr);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

phSipServerMgr

The handle of the The handle of the *SIPServerMgr*.

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrSetEvHandler()

DESCRIPTION

Sets the Server Transport event handler functions of the application layer.

SYNTAX

```
RV_Status RvSipServerTransportMgrSetEvHandler(
   IN RvSipServerTransportMgrHandle     hServerTransportMgr;
   IN RvSipServerTransportEvHandler*    pEvHandler,
   IN RvUint32                          size);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

pEvHandler

The structure of pointers to the application module layer event handler functions.

size

The size of the event handler structure.

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrLIRAdd()

DESCRIPTION

Adds the given LIR to the list and returns a handle to the list entry.

SYNTAX

```
RvStatus RvSipServerTransportMgrLIRAdd (
  IN  RvSipServerTransportMgrHandle     hServerTransportMgr,
  IN  RvSipServerTransportLocalInterfaceRecord*  pSourceLIR,
  OUT RvSipServerTransportLocalInterfaceRecord**     ppLIR);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

pSourceLIR

A pointer to the LIR to add. The address, port and transport type fields are mandatory.

ppLIR

The returned pointer to the list entry which contains the added LIR.

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrLIRFindByAddress()

DESCRIPTION

Searches for a list entry which holds an LIR based on the given key.

SYNTAX

```
RvStatus RvSipServerTransportMgrLIRFindByAddress(
    IN  RvSipServerTransportMgrHandle   hServerTransportMgr,
    IN  RvSipTransportAddr*                          pKey,
    OUT RvSipServerTransportLocalInterfaceRecord**   ppLIR);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

pKey

A pointer to search address key. The address, port and transport type fields are mandatory.

ppLIR

The returned pointer to the list entry which contains the LIR. Otherwise, NULL is returned

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrLIRRemove()

DESCRIPTION

Removes the given LIR list entry. The pLIR pointer was retrieved by one of the following API functions:

- RvSipServerTransportMgrLIRFindByAddress()
- RvSipServerTransportMgrLIRAdd()
- RvSipServerTransportMgrLIRGetFirst()
- RvSipServerTransportMgrLIRGetNext()

SYNTAX

```
RvStatus RvSipServerTransportMgrLIRRemove(
    IN RvSipServerTransportMgrHandle    hServerTransportMgr,
    IN RvSipServerTransportLocalInterfaceRecord*      pLIR);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

pLIR

The pointer of the LIR to be remove by the function.

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrLIRGetFirst()

DESCRIPTION

Returns the first LIR entry related to the *eTransportType* transport type.

SYNTAX

```
RvStatus RvSipServerTransportMgrLIRGetFirst(
    IN RvSipServerTransportMgrHandle    hServerTransportMgr,
    IN RvSipTransport                        eTransportType,
    OUT RvSipServerTransportLocalInterfaceRecord**   ppLIR);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

eTransportType

The transport type.

ppLIR

A pointer to the list entry which contains the first LIR related to the specified transport type. NULL is returned in case the list is empty.

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrLIRGetNext()

DESCRIPTION

Returns the next LIR list element to *pCurLIR*. *pCurLIR* is a pointer to the list element that was retrieved by one of the following API functions:

- RvSipServerTransportMgrLIRGetFirst()
- RvSipServerTransportMgrLIRFindByAddress()

SYNTAX

```
RvStatus RvSipServerTransportMgrLIRGetNext(
  IN  RvSipServerTransportMgrHandle     hServerTransportMgr,
  IN  RvSipServerTransportLocalInterfaceRecord*     pCurLIR,
  OUT RvSipServerTransportLocalInterfaceRecord**    ppLIR);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

pCurLIR

The handle Current LIR list element.

ppLIR

A pointer to the memory where the handle to the found LIR will be stored by the function. Otherwise, NULL is returned.

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrGetNumOfLocalAddress()

DESCRIPTION

Returns the amount of local addresses related to the specified *eTransportType* transport.

SYNTAX

```
RvInt32 RvSipServerTransportMgrGetNumOfLocalAddress(
    IN RvSipServerTransportMgrHandle    hServerTransportMgr,
    IN RvSipTransport                   eTransportType);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

eTransoprtType

The transport type to return its related amount of local addresses.

RETURN VALUES

Returns the amount of local addresses related to the specified transport.

RvSipServerTransportMgrGetNumOfDomains()

DESCRIPTION

Returns the amount of domains that the SIP Server is responsible for (SIP Server domain list).

SYNTAX

```
RvInt32 RvSipServerTransportMgrGetNumOfDomains(
    IN RvSipServerTransportMgrHandle   hServerTransportMgr);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

RETURN VALUES

Returns the amount of domains that the SIP Server is responsible for.

RvSipServerTransportMgrDomainAdd()

DESCRIPTION

Adds the specified domain to the SIP Server domain list.

SYNTAX

```
RvStatus RvSipServerTransportMgrDomainAdd(
    IN RvSipServerTransportMgrHandle    hServerTransportMgr,
    IN const RvChar*                    szDomain);
```

PARAMETERS

hServerTransportMgr

The handle to the *ServerTransportMgr*.

szDomain

The domain to be added.

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrDomainRemove()

DESCRIPTION

Removes the specified domain from the SIP Server Domain list.

SYNTAX

```
RvStatus RvSipServerTransportMgrDomainRemove(
    IN RvSipServerTransportMgrHandle    hServerTransportMgr,
    IN RvChar*                          szDomain);
```

PARAMETERS

RvSipServerTransportMgrHandle

The handle to the *ServerTransportMgr*.

szDomain

The domain to be removed.

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrDomainGetFirst()

DESCRIPTION

Returns the first element of the SIP Server domain list.

SYNTAX

```
RvStatus RvSipServerTransportMgrDomainGetFirst(
    IN RvSipServerTransportMgrHandle    hServerTransportMgr,
    OUT const RvChar**                  pszDomain);
```

PARAMETERS

RvSipServerTransportMgrHandle

The handle to the *ServerTransportMgr*.

psDomain

The returned address to the first element of the SIP Server domain list. NULL is returned in case the list is empty.

RETURN VALUES

Returns RvStatus.

RvSipServerTransportMgrDomainGetNext()

DESCRIPTION

Returns the next element to the *pBaseDomain* of the SIP Server domain list.

SYNTAX

```
RvStatus RvSipServerTransportMgrDomainGetNext(
   IN  RvSipServerTransportMgrHandle    hServerTransportMgr,
   IN  const RvChar*                    pBaseDomain,
   OUT const RvChar**                   pszDomain);
```

PARAMETERS

RvSipServerTransportMgrHandle

The handle to the *ServerTransportMgr*.

pBaseDomain

The current domain list element.

psDomain

The returned address to the next element of the SIP Server domain list. NULL is returned in case the given domain was not found.

RETURN VALUES

Returns RvStatus.

SIP Stack Transport Functions

The Transport functions are:

- RvSipTransportMgrCreateConnection()
- RvSipTransportMgrLocalAddressSetIpTosSockOption()
- RvSipTransportMgrLocalAddressGetIpTosSockOption()
- RvSipTransportConvertStringToIp()
- RvSipTransportConvertIpToString()
- RvSipTransportInjectMsg()
- RvSipTransportConnectionInit()
- RvSipTransportConnectionConnect()
- RvSipTransportConnectionTerminate()
- RvSipTransportConnectionAttachOwner()
- RvSipTransportConnectionDetachOwner()
- RvSipTransportConnectionEnable()
- RvSipTransportConnectionDisable()
- RvSipTransportConnectionIsEnabled()
- RvSipTransportConnectionGetCurrentState()
- RvSipTransportConnectionGetCurrentTlsState()
- RvSipTransportConnectionIsTcpClient() **Deprecated**
- RvSipTransportConnectionGetNumOfOwners()
- RvSipTransportConnectionGetTransportType()
- RvSipTransportConnectionGetLocalAddress()
- RvSipTransportConnectionGetRemoteAddress()
- RvSipTransportConnectionTlsHandshake()
- RvSipTransportConnectionTlsRenegotiate()
- RvSipTransportConnectionGetAppHandle()
- RvSipTransportConnectionSetAppHandle()
- RvSipTransportConnectionTlsGetEncodedCert()
- RvSipTransportConnectionSetIpTosSockOption()
- RvSipTransportConnectionGetIpTosSockOption()
- RvSipTransportTlsEngineConstruct()
- RvSipTransportTlsEngineAddCertificateToChain()
- RvSipTransportTlsEngineAddTrustedCA()
- RvSipTransportTlsEncodeCert()

- RvSipTransportTlsGetCertVerificationError()
- RvSipTransportTlsEngineCheckPrivateKey()
- RvSipTransportConnectionSetIpTosSockOption()
- RvSipTransportConnectionGetIpTosSockOption()
- RvSipTransportGetIPv4LocalAddressByIndex()
- RvSipTransportGetIPv6LocalAddress()
- RvSipTransportGetNumOfIPv4LocalAddresses()
- RvSipTransportSendObjectEvent()
- RvSipTransportConnectionEnableConnByAlias()
- RvSipTransportConnectionGetAlias()
- RvSipTransportConnectionTlsGetUnderlyingSsl()
- RvSipTransportConnectionTlsRenegotiate()
- RvSipTransportMgrLocalAddressCloseSocket()
- RvSipTransportMgrLocalAddressGetAppHandle()
- RvSipTransportMgrLocalAddressGetConnection()
- RvSipTransportMgrLocalAddressGetSockAddrType()
- RvSipTransportMgrLocalAddressSetAppHandle()
- RvSipTransportTlsEngineGetUnderlyingCtx()
- RvSipTransportTlsGetSubjectAltDNS()

RvSipTransportMgrCreateConnection()

DESCRIPTION

Constructs a new un-initialized *connection* and attaches the supplied owner to the *connection*. The owner's event handlers structure is saved with the *connection* owner. The new *connection* assumes the IDLE state. Calling the RvSipTransportConnectionInit() function in this state will initialize the *connection* and will cause the *connection* to move to the READY state.

Note This function does not connect the *connection*. To connect the *connection*, you must first initialize it.

SYNTAX

```
RvStatus RvSipTransportMgrCreateConnection(
   IN  RvSipTransportMgrHandle              hTransportMgr,
   IN  RvSipTransportConnectionOwnerHandle  hOwner,
   IN  RvSipTransportConnectionEvHandlers   *pEvHandlers,
   IN  RvInt32                              sizeofEvHandlers
   OUT RvSipTransportConnectionHandle       *phConn);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hOwner

A handle to the *connection* owner.

pEvHanders

The event handlers structure for this *connection* owner.

sizeofEvHandlers

The size of the event handler structure.

phConn

The handle to the newly created *connection*.

RETURN VALUES

Returns RvStatus.

RvSipTransportMgrLocalAddressSetIpTosSockOption()

DESCRIPTION

Sets the IP_TOS socket option when the value is in decimal form. Note that the option does not provide QoS functionality in operation systems that support a more powerful DSCP mechanism in place of the previous TOS byte mechanism. The function can be called any time during the address life cycle.

SYNTAX

```
RvStatus RvSipTransportMgrLocalAddressSetIpTosSockOption(
    IN RvSipTransportLocalAddrHandle    hLocalAddr,
    IN RvInt32                          typeOfService);
```

PARAMETERS

hLocalAddr

The handle to the local address to be updated.

typeOfService

The number to be set as a TOS byte value.

RETURN VALUES

Returns RvStatus.

RvSipTransportMgrLocalAddressGetIpTosSockOption()

DESCRIPTION

Gets the value of the IP_TOS option that is set for the socket, which serves the specified local address.

SYNTAX

```
RvStatus RvSipTransportMgrLocalAddressGetIpTosSockOption(
    IN RvSipTransportLocalAddrHandle    hLocalAddr,
    IN RvInt32                          *pTypeOfService);
```

PARAMETERS

hLocalAddr

The handle to the local address to be updated.

pTypeOfService

A pointer to the memory where the option value will be stored.

RETURN VALUES

Returns RvStatus.

RvSipTransportConvertStringToIp()

DESCRIPTION

Converts an IP address from string format to binary format.

SYNTAX

```
RvStatusRvSipTransportConvertStringToIp(
    IN  RvSipTransportMgrHandle      hTransportMgr,
    IN  RvChar*                      strIpAsString,
    IN  RvSipTransportAddressType    eAddressType,
    OUT RvUint8*                     pIpAsBinary);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

strIpAsString

A NULL terminated string representing an IP address (d.d.d.d for IPv4, x:x:x:x:x:x:x:x for IPv6).

eAddressType

The type of address (IPv6 or IPv4).

pIpAsBinary

The IP address represented in a binary format (16 bytes for IPv6, 4 bytes for IPv4).

RETURN VALUES

Returns RvStatus.

RvSipTransportConvertIpToString()

DESCRIPTION

Converts an IP address from binary format to string format.

SYNTAX

```
RvStatus RvSipTransportConvertIpToString(
    IN  RvSipTransportMgrHandle      hTransportMgr,
    IN  RvUint8*                     pIpAsBinary,
    IN  RvSipTransportAddressType    eAddressType,
    IN  RvInt32                      stringLen,
    OUT RvChar*                      strIpAsString);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

pIpAsBinary

The IP address represented in a binary format (16 bytes for IPv6, 4 bytes for IPv4).

stringLen

The size of the buffer.

eAddressType

The type of address (IPv6 or IPv4).

strIpAsString

A NULL terminated string representing an IP address (d.d.d.d for IPv4, x:x:x:x:x:x:x:x for IPv6).

RETURN VALUES

Returns RvStatus.

RvSipTransportInjectMsg()

DESCRIPTION

Enables the application to "inject" a message into the Stack. The Stack will handle the message as if it was received from the network. The message may be given as a string or as a message object. You can optionally supply a local and remote addresses for this message. Suppling a local address is useful in the case of multihomed host, when you want to simulate a message that is received from a specific network card. For Request messages, this will cause the response to be sent from the same card. If you supply a remote address, this address will be set as the received parameter of the Via header for incoming requests. Responses will then be sent to this address. If you want to simulate a message that was received on a specific *connection*, you can supply a *connection* handle. In this case, the local and remote addresses will be taken from the *connection* and the *pAddressInfo* will be ignored.

Note If you do not wish to supply addresses, you can set a NULL value in the *pAddressInfo* and the *hConn* parameters.

SYNTAX

```
RvStatus RvSipTransportInjectMsg(
    IN RvSipTransportMgrHandle          hTransportMgr,
    IN RvChar                           *pMsgBuffer,
    IN RvUint32                         totalMsgLength,
    IN RvSipMsgHandle                   hMsg,
    IN RvSipTransportConnectionHandle   hConn,
    IN RvSipTransportMsgAddrCfg         *pAddressInfo);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

pMsgBuffer

The "injected" message in string format.

totalMsgLength

The total length of the message given in *pMsgBuffer*.

hMsg

The handle to the “injected” message in message object format.

hConn

The *connection* handle to which the message is “injected”.

pAddressInfo

A structure that contains the local and remote addresses.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionInit()

DESCRIPTION

Initializes a *connection* with all needed configuration parameters found in the RvSipTransportConnectionCfg structure. You can call this function only in the IDLE state. This function causes the *connection* to move to the READY state. The initialized *connection* is inserted into the *connection* hash and therefore can be used by any persistent Stack object.

Note This function does not connect the *connection*. The *connection* is connected when a Stack object uses it for sending a message, or if you specifically call the RvSipTransportConnectionConnect() function. In both cases, the *connection* will assume the CONNECTING and then TCP_CONNECTED or SCTP_CONNECTED states.

SYNTAX

```
RvStatus RvSipTransportConnectionInit(
    IN RvSipTransportConnectionHandle    hConn,
    IN RvSipTransportConnectionCfg       *pCfg)
    IN RvInt32                           sizeofCfg;
```

PARAMETERS

hConn

The handle to the *connection* to be initialized.

pCfg

The configuration to use when initializing the *connection*.

sizeofCfg

The size of the configuration structure.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionConnect()

DESCRIPTION

Connects a *connection*. You can call this function only in the READY state. Calling this function will cause the *connection* to move to the CONNECTING state. The *connection* will move to the TCP_CONNECTED or SCTP_CONNECTED state when an indication that the *connection* was successfully connected is received from the network.

SYNTAX

```
RvStatus RvSipTransportConnectionConnect(
    IN RvSipTransportConnectionHandle    hConn);
```

PARAMETERS

hConn

The handle to the *connection* to connect.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionTerminate()

DESCRIPTION

The function behavior depends on the *connection* state. If the *connection* is in the TCP_CONNECTED, TLS_CONNECTED or SCTP_CONNECTED state, the *connection* will start a normal disconnection process. TCP and SCTP *connections* will move to the CLOSING state. TLS *connections* will move to the TLS_CLOSE_SEQUENSE_STARTED state. For all other states, the *connection* will close its internal socket if the socket was opened, and will terminate. After termination the *connection* will assume the TERMINATED state.

Note If the *connection* has messages that it is about to send, these messages will be lost. It is therefore not recommended to use this function. If you no longer need this *connection*, call the RvSipTransportConnectionDetachOwner() function. The *connection* will be closed only when the last owner is detached. This means that if the *connection* is still being used by other Stack objects, it will not be closed until these objects detach from it.

SYNTAX

```
RvStatus RvSipTransportConnectionTerminate(
    IN RvSipTransportConnectionHandle    hConn);
```

PARAMETERS

hConn

The handle to the *connection* to be terminated.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionAttachOwner()

DESCRIPTION

Attaches a new owner to the supplied *connection* with a set of callback functions that will be used to notify this owner about *connection* events. You can use this function only with client *connections* and if the *connection* is connected, or in the process of being connected. You cannot attach an owner to a *connection* that started its disconnection process.

Note The *connection* will not disconnect as long as it has owners attached to it.

SYNTAX

```
RvStatus RvSipTransportConnectionAttachOwner(
  IN RvSipTransportConnectionHandle        hConn,
  IN RvSipTransportConnectionOwnerHandle   hOwner,
  IN RvSipTransportConnectionEvHandlers    *pEvHandlers),
  IN RvInt32                               sizeOfEvHandlers;
```

PARAMETERS

hConn

The handle to the *connection*.

hOwner

The owner handle.

pEvHanders

The event handlers structure for this *connection* owner.

sizeOfEvHandlers

The size of the event handler structure.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionDetachOwner()

DESCRIPTION

Detaches an owner from the supplied *connection*. If the *connection* is left with no other owners, it will be closed. If the same owner is attached to a *connection* more than once, the first matching owner will be removed.

Note After detaching from a *connection*, you will stop getting *connection* events and you must not use the *connection* any longer.

SYNTAX

```
RvStatus RvSipTransportConnectionDetachOwner(
    IN RvSipTransportConnectionHandle         hConn,
    IN RvSipTransportConnectionOwnerHandle    hOwner);
```

PARAMETERS

hConn

The handle to the *connection*.

hOwner

The handle to the owner to detach from the *connection*.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionEnable()

DESCRIPTION

Inserts a *connection* into the hash so that persistent objects will be able to use it.

Note When ever a *connection* is initialized with the function RvSipTransportConnectionInit() it is automatically inserted to the hash.

SYNTAX

```
RvStatus RvSipTransportConnectionEnable(
    IN RvSipTransportConnectionHandle    hConn);
```

PARAMETERS

hConn

The handle to the *connection*.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionDisable()

DESCRIPTION

Removes a *connection* from the hash so that persistent objects will not be able to use it. Objects that are already using the *connection* (are in the *connection* owners list) will continue to use the *connection*. However, other objects will not be able to use the *connection* as long as the *connection* is disabled.

Note To insert the *connection* back to the hash, use RvSipTransportConnectionEnable().

SYNTAX

```
RvStatus RvSipTransportConnectionDisable(
    IN RvSipTransportConnectionHandle    hConn);
```

PARAMETERS

hConn

The handle to the *connection*.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionIsEnabled()

DESCRIPTION

Returns whether or not the *connection* is enabled (in the hash). An enabled *connection* is a *connection* that can be used by persistent objects.

SYNTAX

```
RvStatus RvSipTransportConnectionIsEnabled(
    IN  RvSipTransportConnectionHandle    hConn,
    OUT RvBool                            *pbIsEnabled);
```

PARAMETERS

hConn

The handle to the *connection*.

pbIsEnabled

RV_TRUE if the *connection* is enabled. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionGetCurrentState()

DESCRIPTION

Retrieves the current state of the *connection*.

SYNTAX

```
RvStatus RvSipTransportConnectionGetCurrentState(
    IN  RvSipTransportConnectionHandle    hConn,
    OUT RvSipTransportConnectionState     *peState);
```

PARAMETERS

hConn

The handle to the *connection*.

peState

The current state of the *connection*.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionGetCurrentTlsState()

DESCRIPTION

Retrieves the current TLS state of the *connection*.

SYNTAX

```
RvStatus RvSipTransportConnectionGetCurrentTlsState(
    IN  RvSipTransportConnectionHandle       hConn,
    OUT RvSipTransportConnectionTlsState*    peState)
```

PARAMETERS

hConn

The handle to the *connection*.

peState

The current TLS state of the *connection*.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionIsTcpClient()

DESCRIPTION

This function is deprecated and kept for backward compatibility only.

SYNTAX

```
RvStatus RvSipTransportConnectionIsTcpClient(
    IN  RvSipTransportConnectionHandle    hConn,
    OUT RvBool                            *pbIsClient);
```

PARAMETERS

hConn

The handle to the *connection*.

pbIsClient

RV_TRUE if the *connection* is a TCP client. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionGetNumOfOwners()

DESCRIPTION

Retrieves the number of owners currently attached to the *connection*.

SYNTAX

```
RvStatus RvSipTransportConnectionGetNumOfOwners(
    IN  RvSipTransportConnectionHandle    hConn,
    OUT RvInt32                           *pNumOfOwners);
```

PARAMETERS

hConn

The handle to the *connection*.

pNumOfOwners

The number of *connection* owners.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionGetTransportType()

DESCRIPTION

Retrieves the *connection* transport (TCP, TLS or SCTP).

SYNTAX

```
RvStatus RvSipTransportConnectionGetTransportType(
    IN  RvSipTransportConnectionHandle    hConn,
    OUT RvSipTransport                    *peTransport);
```

PARAMETERS

hConn

The handle to the *connection*.

peTransport

The *connection* transport.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionGetLocalAddress()

DESCRIPTION

Retrieves the local address of the *connection*. The local address includes the local IP, local port and the address type (IPv4 or IPv6).

SYNTAX

```
RvStatus RvSipTransportConnectionGetLocalAddress(
    IN  RvSipTransportConnectionHandle    hConn,
    OUT RvChar                            *strAddress,
    OUT RvUint16                          *pPort,
    OUT RvSipTransportAddressType         *peAddressType);
```

PARAMETERS

hConn

The handle to the *connection*.

pAddress

A previously allocated buffer to where the local address will be copied. The buffer should have a minimum size of 51 (RVSIP_TRANSPORT_LEN_STRING_IP).

pPort

The local port.

peAddressType

The local address type, IPV4 or IPv6.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionGetRemoteAddress()

DESCRIPTION

Retrieves the remote address of the *connection*. The remote address includes the remote IP, remote Port and the remote address type (IPv4 or IPv6).

SYNTAX

```
RvStatus RvSipTransportConnectionGetRemoteAddress(
    IN  RvSipTransportConnectionHandle    hConn,
    OUT RvChar                            *strAddress,
    OUT RvUint16                          *pPort,
    OUT RvSipTransportAddressType         *peAddressType);
```

PARAMETERS

hConn

The handle to the *connection*.

srtAddress

A previously allocated buffer to where the remote address will be copied. The buffer should have a minimum size of 51 (RVSIP_TRANSPORT_LEN_STRING_IP).

pPort

The remote port.

peAddressType

The remote address type (IPV4 or IPv6).

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionTlsHandshake()

DESCRIPTION

Starts TLS negotiation on a *connection*.

SYNTAX

```
RvStatus RvSipTransportConnectionTlsHandshake(
    IN RvSipTransportConnectionHandle
                                    hConnection,
    IN RvSipTransportTlsEngineHandle
                                    hEngine,
    IN RvSipTransportTlsHandshakeSide
                                    eHandshakeSide,
    IN RvSipTransportVerifyCertificateEv
                                    pfnVerifyCertEvHandler);
```

PARAMETERS

hConnection

The *connection* on which to start the handshake.

hEngine

The TLS engine that will be associated with the *connection*. The *connection* will "inherit" the parameters of the engine.

eHandshakeSide

The TLS handshake side that the *connection* will play on the TLS handshake. Using the default enumeration will set the handshake side to Client for TCP clients and Server for TCP servers.

pfnVerifyCertEvHandler

A callback to check certificates that arrived during the handshake.

On the client handshake side, NULL means use the default callback. Valid certificates will be approved and invalid certificates will be rejected, causing a handshake failure. The callback function supplied here overrides that default.

On the server handshake side, NULL means no client certificates. The callback function supplied here will require a client certificate.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionTlsRenegotiate()

DESCRIPTION

Starts TLS renegotiation on a *connection*.

SYNTAX

```
RvStatus RvSipTransportConnectionTlsRenegotiate(
    IN RvSipTransportConnectionHandle    hConnection);
```

PARAMETERS

hConnection

The *connection* on which to start the TLS renegotiation.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionGetAppHandle()

DESCRIPTION

Retrieves the application handle to the *connection*.

SYNTAX

```
RvStatus RvSipTransportConnectionGetAppHandle(
    IN  RvSipTransportConnectionHandle       hConn,
    OUT RvSipTransportConnectionAppHandle    *phAppHandle);
```

PARAMETERS

hConn

The *connection* handle.

phAppHandle

The *connection* application handle.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionSetAppHandle()

DESCRIPTION

Sets the application handle to the *connection*.

SYNTAX

```
RvStatus RvSipTransportConnectionSetAppHandle(
    IN RvSipTransportConnectionHandle       hConn,
    IN RvSipTransportConnectionAppHandle    hAppHandle);
```

PARAMETERS

hConn

The *connection* handle.

hAppHandle

The *connection* application handle.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionTlsGetEncodedCert()

DESCRIPTION

Retrieves a certificate from a *connection* if the allocated buffer is insufficient. The length of the buffer that is needed will be inserted in *pCertLen*.

SYNTAX

```
RvStatus RvSipTransportConnectionTlsGetEncodedCert(
    IN    RvSipTransportConnectionHandle    hConnection,
    INOUT RvInt32                           *pCertLen,
    OUT   RvChar                            *strCert);
```

PARAMETERS

hConnection

The *connection* on which to get the certificate.

pCertLen

The allocated certificate buffer length.

pCertLen

The real size of the certificate in case the buffer was not sufficient.

strCert

The allocated buffer to hold the certificate.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionSetIpTosSockOption()

DESCRIPTION

Sets the IP_TOS socket option for socket, serving the *connection*. Note that the option does not provide QoS functionality in operation systems that support a more powerful DSCP mechanism in place of the previous TOS byte mechanism. The function can be called any time during the address life cycle.

SYNTAX

```
RvStatus RvSipTransportConnectionSetIpTosSockOption(
    IN RvSipTransportConnectionHandle    hConn,
    IN RvInt32                           typeOfService);
```

PARAMETERS

hConn

The handle to the *connection* to be updated.

typeOfService

The number to be set as a TOS byte value.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionGetIpTosSockOption()

DESCRIPTION

Gets the value of the IP_TOS option that is set for the socket, which serves the specified *connection*.

SYNTAX

```
RvStatus RvSipTransportConnectionGetIpTosSockOption(
    IN RvSipTransportConnectionHandle    hConn,
    IN RvInt32                           *pTypeOfService);
```

PARAMETERS

hConn

The handle to the *connection* to be updated.

pTypeOfService

A pointer to the memory where the option value will be stored.

RETURN VALUES

Returns RvStatus.

RvSipTransportTlsEngineConstruct()

DESCRIPTION

Constructs a TLS engine. A TLS engine is an entity that holds together a number of characteristics related to TLS sessions. When making a TLS handshake, you have to provide an engine. The handshake parameters will be derived from the parameters of the engines. For example, you can create a "TLS client" engine by calling the RvSipTransportTlsEngineAddTrustedCA() function after an engine has been constructed. Once an engine has been constructed, it can be used to perform TLS handshakes. A handshake that uses an engine will "inherit" its TLS characteristics, for example, the TLS version.

SYNTAX

```
RvStatus RvSipTransportTlsEngineConstruct(
    IN  RvSipTransportMgrHandle          hTransportMgr,
    IN  RvSipTransportTlsEngineCfg       *pTlsEngineCfg,
    IN  RvInt32                          sizeofCfg
    OUT RvSipTransportTlsEngineHandle    *phTlsEngine);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

pTlsEngineCfg

A pointer to the configuration structure that holds data for the TLS engine.

sizeofCfg

The size of the configuration structure.

phTlsEngine

The newly created TLS engine.

RETURN VALUES

Returns RvStatus.

RvSipTransportTlsEngineAddCertificateToChain()

DESCRIPTION

Adds a TLS certificate to the chain of certificates. The engine holds a chain of certificates needed for its approval (usually ending with a self-signed certificate). The engine will display the chain of certificates during handshakes, in which it is required to present certificates.

SYNTAX

```
RvStatus RvSipTransportTlsEngineAddCertificateToChain(
    IN RvSipTransportMgrHandle          hTransportMgr,
    IN RvSipTransportTlsEngineHandle    hTlsEngine,
    IN RvChar                           *strCert,
    IN RvInt32                          certLen);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hTlsEngine

The handle to the TLS engine.

strCert

The certificate encoded as ASN.1 string representation

certLen

The length of the certificate.

RETURN VALUES

Returns RvStatus.

RvSipTransportTlsEngineAddTrustedCA()

DESCRIPTION

Adds a trusted Certificate Authority (CA) to an engine. After using this function, the engine will approve all certificates issued by the CA. A CA is an entity that issues certificates. Most TLS clients on the net trust one or more CAs and approve only certificates that were issued by those CAs. After adding a trusted CA to an engine, you can use it as a "TLS client" engine and use that *connection* on handshakes in which you request the other side of the *connection* to display its certificates.

SYNTAX

```
RvStatus RvSipTransportTlsEngineAddTrustedCA(
    IN RvSipTransportMgrHandle          hTransportMgr,
    IN RvSipTransportTlsEngineHandle    hTlsEngine,
    IN RvChar                           *strCert,
    IN RvInt32                          certLen);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hTlsEngine

The handle to the TLS engine.

strCert

The certificate encoded as ASN.1 string representation.

certLen

The length of the certificate.

RETURN VALUES

Returns RvStatus.

RvSipTransportTlsEncodeCert()

DESCRIPTION

Encodes a certificate to a buffer in DER(ASN.1) format.

SYNTAX

```
RvStatus RvSipTransportTlsEncodeCert(
    IN    RvSipTransportTlsCertificate    hCert,
    INOUT RvInt32                         *pCertLen,
    OUT   RvChar                          *strCert);
```

PARAMETERS

hCert

The certificate to encode.

pCertLen

The buffer length.

strCert

The certificate encoded into Asn.1 format.

pCertLen

The length of the certificate (in bytes).

RETURN VALUES

Returns RvStatus.

RvSipTransportTlsGetCertVerificationError()

DESCRIPTION

Retrieves an error string in the verification callback.

SYNTAX

```
RvChar *RvSipTransportTlsGetCertVerificationError(
    IN  RvSipTransportTlsCertificate    hCert,
    OUT RvChar                          **strError);
```

PARAMETERS

hCert

The handle to the certificate.

strError

The error string.

RETURN VALUES

Returns RvStatus.

RvSipTransportTlsEngineCheckPrivateKey()

DESCRIPTION

Checks the consistency of a private key with the corresponding certificate loaded into ctx. If more than one key/certificate pair (RSA/DSA) is installed, the last installed item will be checked. If, for example, the last item was a RSA certificate or key, the RSA key/certificate pair will be checked. This is a utility function for the application to make sure that the key and certificate were loaded correctly into the engine.

SYNTAX

```
RvStatus RvSipTransportTlsEngineCheckPrivateKey(
    IN RvSipTransportMgrHandle          hTransportMgr,
    IN RvSipTransportTlsEngineHandle    hTlsEngine);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hTlsEngine

The TLS engine.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionSetIpTosSockOption()

DESCRIPTION

Sets the IP_TOS socket option for the socket serving the *connection*. The option value is in decimal form. Note that the option does not provide QoS functionality in operation systems that support a more powerful DSCP mechanism in place of the previous TOS byte mechanism. The function can be called any time during the address life cycle.

SYNTAX

```
RvStatus RvSipTransportConnectionSetIpTosSockOption(
    IN RvSipTransportConnectionHandle    hConn,
    IN RvInt32                           typeOfService);
```

PARAMETERS

hConn

The handle to the *connection* to be updated.

typeOfService

The number to be set as a TOS byte value.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionGetIpTosSockOption()

DESCRIPTION

Gets the value of the IP_TOS option that is set for the socket, which serves the specified *connection*.

SYNTAX

```
RvStatus RvSipTransportConnectionGetIpTosSockOption(
    IN RvSipTransportConnectionHandle    hConn,
    IN RvInt32                           *pTypeOfService);
```

PARAMETERS

hConn

The handle to the *connection* to be updated.

pTypeOfService

A pointer to the memory where the option value will be stored.

RETURN VALUES

Returns RvStatus.

RvSipTransportGetIPv4LocalAddressByIndex()

DESCRIPTION

Retrieves the local address by index. This function is used when the SIP Stack was initialized with IPv4 local address of 0, and therefore listens on several distinct local addresses. To know how many local addresses are available by this function, call the RvSipTransportGetNumOfIPv4LocalAddresses() function. If, for example, this function returns five, you can call RvSipTransportGetIPv4LocalAddressByIndex() with indexes from 0 to 4.

Note The IPv4 address requires 4-BYTEs of memory. This is the same as an unsigned int (RvUint32). This function requires *pLocalAddr* to be a pointer to a 4-BYTE allocated memory. It can also be a pointer to RvUint32 with an appropriate casting.

SYNTAX

```
RvStatus RvSipTransportGetIPv4LocalAddressByIndex(
    IN  RvSipTransportMgrHandle    hTransportMgr,
    IN  RvUint                     index,
    OUT RvUint8                    *pLocalAddr)
```

PARAMETERS

pTransportMgr

A pointer to the *TransportMgr*.

index

The index for the local address to retrieve.

pLocalAddr

A pointer to a 4-BYTE memory space to be filled with the selected local address.

RETURN VALUES

Returns RvStatus.

RvSipTransportGetIPv6LocalAddress()

DESCRIPTION

Retrieves the local address that was actually open for listening when the SIP Stack was initiated with local address 0:0:0:0:0:0:0:0.

Note The IPv6 address requires 16-BYTEs of memory. This function requires *pLocalAddr* to be a pointer to a 16-BYTE allocated memory.

SYNTAX

```
RvStatus RvSipTransportGetIPv6LocalAddress(
    IN  RvSipTransportMgrHandle    hTransportMgr,
    OUT RvUint8*                   pLocalAddr)
```

PARAMETERS

pTransportMgr

A pointer to the *TransportMgr*.

pLocalAddr

A pointer to a 16-BYTE memory space to be filled with the selected local address.

RETURN VALUES

Returns RvStatus.

RvSipTransportGetNumOfIPv4LocalAddresses()

DESCRIPTION

Returns the number of local addresses to which the SIP Stack listens.

SYNTAX

```
RvStatus RvSipTransportGetNumOfIPv4LocalAddresses(
    IN  RvSipTransportMgrHandle    hTransportMgr,
    OUT RvUint32                   *pNumberOfAddresses)
```

PARAMETERS

pTransportMgr

A pointer to the *TransportMgr*.

pNumberOfAddresses

The number of local addresses for which the SIP Stack listens.

RETURN VALUES

Returns RvStatus.

RvSipTransportSendObjectEvent()

DESCRIPTION

Sends an event through the event queue.

SYNTAX

```
RvStatus RvSipTransportSendObjectEvent(
    IN RvSipTransportMgrHandle             hTransportMgr,
    IN void*                               pObj,
    IN RvSipTransportObjEventInfo*         pEventInfo,
    IN RvInt32                             reason,
    IN RvSipTransportObjectEventHandler    func)
```

PARAMETERS

hTransportMgr

The *TransportMgr* handle.

pObj

A pointer to the object to be terminated.

pEventInfo

A pointer to an allocated un-initialized structure for queueing object events.

reason

The event reason.

func

The event callback function. This function will be called when the event will be popped from the event queue.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionEnableConnByAlias()

DESCRIPTION

Allows a server *connection* to be reused in the future, by inserting the *connection* into the *connections* hash table. When an incoming request has an alias parameter in its top-most Via header, the RvSipTransportConnectionServerReuseEv() callback is called. In this callback, the application should authorize this *connection*, and if authorized, should use this function to allow the server *connection* to be reused when sending future requests. The *connection* is identified by an alias string, given in the top-most Via header.

(This function is different from RvSipTransportConnectionEnable() since it uses the alias name of the *connection*, and not the remote address).

SYNTAX

```
RvStatus RvSipTransportConnectionEnableConnByAlias(
    IN RvSipTransportConnectionHandle    hConn);
```

PARAMETERS

hConn

The handle to the *connection* that can be reused.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionGetAlias()

DESCRIPTION

Retrieves the *connection* alias string. The function always retrieves the alias string length. If an allocated buffer is also given, and *allocatedBufferLen* is less then zero, the alias string will be copied to the buffer.

SYNTAX

```
RvStatus RvSipTransportConnectionConnect(
   IN  RvSipTransportConnectionHandle    hConn,
   IN  RvInt32                           allocatedBufferLen,
   OUT RvInt32                           *pAliasLength,
   OUT RvChar                            *pBuffer);
```

PARAMETERS

hConn

This function retrieves the *connection* alias string. The function always retrieves the alias string length. If an allocated buffer is also given, and *allocatedBufferLen* is greater then zero, the alias string will be copied to the buffer.

allocatedBufferLen

The length of the given allocated buffer.

pAliasLength

The length of the alias string. (Zero if the string does not exist.)

pBuffer

The buffer to be filled with the alias string.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionTlsGetUnderlyingSsl()

DESCRIPTION

Gets the pointer to the SSL object that the TLS session uses. This pointer can be used to apply direct OpenSSL API functions on the session. Once the session was exposed, the application can change the session settings, but the application is not allowed to interfere with the I/O operations of the session.

SYNTAX

```
RvStatus RvSipTransportConnectionTlsGetUnderlyingSsl(
    IN  RvSipTransportConnectionHandle    hConn,
    OUT void**                            pUnderlyingSSL);
```

PARAMETERS

hConn

The handle to the *connection*.

pUnderlyingSSL

The underlying SSL session of the *connection*.

RETURN VALUES

Returns RvStatus.

RvSipTransportConnectionTlsRenegotiate()

DESCRIPTION

Starts TLS renegotiation on a *connection*.

SYNTAX

```
RvStatus RvSipTransportConnectionTlsRenegotiate(
    IN RvSipTransportConnectionHandle    hConn);
```

PARAMETERS

hConn

The *connection* on which to start renegotiation.

RETURN VALUES

Returns RvStatus.

RvSipTransportMgrLocalAddressCloseSocket()

DESCRIPTION

Closes the socket of the given local address object.

SYNTAX

```
RvStatus RvSipTransportMgrLocalAddressCloseSocket(
    IN RvSipTransportLocalAddrHandle    hLocalAddr);
```

PARAMETERS

hLocalAddr

The handle to the local address object whose socket should be closed.

RETURN VALUES

Returns RvStatus.

RvSipTransportMgrLocalAddressGetAppHandle()

DESCRIPTION

Returns an application handle to the local address object. You set this handle in the SIP Stack using the RvSipTransportMgrLocalAddressSetAppHandle() function.

SYNTAX

```
RvStatus RvSipTransportMgrLocalAddressGetAppHandle(
    IN RvSipTransportLocalAddrHandle    hLocalAddr);
```

PARAMETERS

hLocalAddr

The handle to the local address object.

RETURN VALUES

Returns RvStatus.

RvSipTransportMgrLocalAddressGetConnection()

DESCRIPTION

Gets the handle to the listening *connection* bound to the local address object of TCP, TLS or SCTP type.

SYNTAX

```
RvStatus RvSipTransportMgrLocalAddressGetConnection(
    IN  RvSipTransportLocalAddrHandle     hLocalAddr,
    OUT RvSipTransportConnectionHandle    *phConn);
```

PARAMETERS

hLocalAddr

The handle to the local address object.

phConn

A pointer to the memory where the handle will be stored.

RETURN VALUES

Returns RvStatus.

RvSipTransportMgrLocalAddressGetSockAddrType()

DESCRIPTION

Gets type of addresses (IPv4 or IPv6) that the local address socket supports.

SYNTAX

```
RvStatus RvSipTransportMgrLocalAddressGetSockAddrType (
    IN  RvSipTransportLocalAddrHandle    hLocalAddr,
    OUT RvSipTransportAddressType*       peSockAddrType);
```

PARAMETERS

hLocalAddr

The handle to the local address object.

peSockAddrType

The type of address to which the socket is bound.

RETURN VALUES

Returns RvStatus.

RvSipTransportMgrLocalAddressSetAppHandle()

DESCRIPTION

Sets an application handle to the local address object. You get this handle in the SIP Stack using the RvSipTransportMgrLocalAddressGetAppHandle() function.

SYNTAX

```
RvStatus RvSipTransportMgrLocalAddressSetAppHandle (
    IN RvSipTransportLocalAddrHandle       hLocalAddr,
    IN RvSipTransportLocalAddrAppHandle    hAppLocalAddr);
```

PARAMETERS

hLocalAddr

The handle to the local address object.

hAppLocalAddr

The application handle to the local address object.

RETURN VALUES

Returns RvStatus.

RvSipTransportTlsEngineGetUnderlyingCtx()

DESCRIPTION

Gets the pointer to the CTX object that the TLS engine uses. This pointer can be used to apply direct OpenSSL API functions on the engine. Once the CTX was exposed, the application can change the engine settings, but the application is not allowed to interfere with the I/O operation of the engine (for example, accept a session with the engine).

SYNTAX

```
RvStatus RvSipTransportTlsEngineGetUnderlyingCtx (
    IN  RvSipTransportMgrHandle          hTransportMgr,
    IN  RvSipTransportTlsEngineHandle    hTlsEngine,
    OUT void**                           pUnderlyingCTX);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hTlsEngine

The handle to the TLS engine.

pUnderlyingCTX

The pointer to the CTX object.

RETURN VALUES

Returns RvStatus.

RvSipTransportTlsGetSubjectAltDNS()

DESCRIPTION

Retrieves a list of the subjectAltNames that are in DNS format from the TLS certificate.

SYNTAX

```
RvStatus RvSipTransportTlsGetSubjectAltDNS (
   IN    RvSipTransportConnectionHandle  hConn,
   INOUT RvChar*                         pBuffer,
   INOUT RvInt32*                        pBufferLen,
   OUT   RvInt32*                        pNumOfNamesInList);
```

PARAMETERS

hConn

The TLS *connection*.

pBuffer

A buffer allocated by the application. The function fills this buffer with the list of DNS names, separated by zeros.

pBufferLen

A pointer to the size of the *pBuffer* buffer. The *pBufferLen* parameter will be filled with the size of the list of DNS names. If the size of the DNS name list is greater than *pBufferLen, pBuffer* will not be filled.

pNumOfNamesInList

A pointer the number of SubjAltNames in the output list.

RETURN VALUES

Returns RvStatus.

DNS TRANSPORT FUNCTIONS

The Transport DNS functions are:

- RvSipTransportDNSListPushSrvElement()
- RvSipTransportDNSListRemoveTopmostSrvElement()
- RvSipTransportDNSListGetSrvElement()
- RvSipTransportDNSListPopSrvElement()
- RvSipTransportDNSListPushHostElement()
- RvSipTransportDNSListRemoveTopmostHostElement()
- RvSipTransportDNSListGetHostElement()
- RvSipTransportDnsGetEnumResult()
- RvSipTransportDNSListPopHostElement()
- RvSipTransportDNSListPushIPElement()
- RvSipTransportDNSListRemoveTopmostIPElement()
- RvSipTransportDNSListGetIPElement()
- RvSipTransportDNSListPopIPElement()
- RvSipTransportGetNumberOfDNSListEntries()
- RvSipTransportDNSListGetUsedHostElement()
- RvSipTransportDNSListGetUsedSRVElement()
- RvSipTransportDNSListSetUsedSRVElement()
- RvSipTransportDNSListSetUsedHostElement()
- RvSipTransportDNSListConstruct()
- RvSipTransportDNSListDestruct()
- RvSipTransportDNSListGetSrvElement()

RvSipTransportDNSListPushSrvElement()

DESCRIPTION

Adds a single SRV element to the head of the SRV names list of the DNS list object.

SYNTAX

```
RvStatus RvSipTransportDNSListPushSrvElement(
    IN RvSipTransportMgrHandle        hTransportMgr,
    IN RvSipTransportDNSListHandle    hDnsList,
    IN RvSipTransportDNSSRVElement    *pSrvElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

pSrvElement

The SRV element structure to be added to the list.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListRemoveTopmostSrvElement()

DESCRIPTION

Removes the topmost SRV element from the SRV elements list of the DNS list object.

SYNTAX

```
RvStatus RvSipTransportDNSListRemoveTopmostSrvElement(
    IN RvSipTransportMgrHandle        hTransportMgr,
    IN RvSipTransportDNSListHandle    hDnsList);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListGetSrvElement()

DESCRIPTION

Retrieves a SRV element from the SRV list of the DNS list object according to the input location.

SYNTAX

```
RvStatus RvSipTransportDNSListGetSrvElement(
    IN    RvSipTransportMgrHandle        hTransportMgr,
    IN    RvSipTransportDNSListHandle    hDnsList,
    IN    RvSipListLocation              location,
    INOUT void                           **pRelative,
    OUT   RvSipTransportDNSSRVElement    *pSrvElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

location

The starting element location.

pRelative

INPUT: The relative SRV element. Used when the location is “next” or “previous”.

OUTPUT: The new relative SRV element.

pSrvElement

The found element.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListPopSrvElement()

DESCRIPTION

Retrieves and removes the topmost SRV name element from the SRV elements list of the DNS list object.

SYNTAX

```
RvStatus RvSipTransportDNSListPopSrvElement(
    IN  RvSipTransportMgrHandle        hTransportMgr,
    IN  RvSipTransportDNSListHandle    hDnsList,
    OUT RvSipTransportDNSSRVElement    *pSrvElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

pSrvElement

The retrieved element.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListPushHostElement()

DESCRIPTION

Adds a host element to the head of the host elements list of the DNS list object.

SYNTAX

```
RvStatus RvSipTransportDNSListPushHostElement(
    IN RvSipTransportMgrHandle              hTransportMgr,
    IN RvSipTransportDNSListHandle          hDnsList,
    IN RvSipTransportDNSHostNameElement     *pHostElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

pHostElement

The host name element structure to be added to the list.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListRemoveTopmostHostElement()

DESCRIPTION

Removes topmost host element from the head of the host elements list of the DNS list object.

SYNTAX

```
RvStatus RvSipTransportDNSListRemoveTopmostHostElement(
    IN RvSipTransportMgrHandle        hTransportMgr,
    IN RvSipTransportDNSListHandle    hDnsList);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListGetHostElement()

DESCRIPTION

Retrieves a host element from the host elements list of the DNS list object according to the input location.

SYNTAX

```
RvStatus RvSipTransportDNSListGetHostElement(
   IN    RvSipTransportMgrHandle             hTransportMgr,
   IN    RvSipTransportDNSListHandle         hDnsList,
   IN    RvSipListLocation                   location,
   INOUT void                                **pRelative,
   OUT   RvSipTransportDNSHostNameElement    *pHostElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

location

The starting element location.

pRelative

INPUT: The relative host name element for the “next” or “previous” locations.
OUTPUT: The new relative host name element.

pHostElement

The found element.

RETURN VALUES

Returns RvStatus.

RvSipTransportDnsGetEnumResult()

DESCRIPTION

Retrieves the result of an ENUM NAPTR query from the DNS list object.

SYNTAX

```
RvStatus RvSipTransportDnsGetEnumResult(
    IN  RvSipTransportMgrHandle        hTransportMgr,
    IN  RvSipTransportDNSListHandle    hDnsList,
    OUT RvChar                         **ppEnumRes);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

ppEnumRes

A pointer to the ENUM string.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListPopHostElement()

DESCRIPTION

Retrieves and removes the topmost host element from the list of host elements in the DNS list object.

SYNTAX

```
RvStatus RvSipTransportDNSListPopHostElement(
    IN  RvSipTransportMgrHandle             hTransportMgr,
    IN  RvSipTransportDNSListHandle         hDnsList,
    OUT RvSipTransportDNSHostNameElement    *pHostElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

pHostElement

The element that was removed from the host list.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListPushIPElement()

DESCRIPTION

Adds single IP address element to the head of the IP addresses list of the DNS list object.

SYNTAX

```
RvStatus RvSipTransportDNSListPushIPElement(
    IN RvSipTransportMgrHandle        hTransportMgr,
    IN RvSipTransportDNSListHandle    hDnsList,
    IN RvSipTransportDNSIPElement     *pIPElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

pIPElement

The IP address element structure to be added to the list.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListRemoveTopmostIPElement()

DESCRIPTION

Removes the topmost element from the head of the DNS list object IP addresses list.

SYNTAX

```
RvStatus RvSipTransportDNSListRemoveTopmostIPElement(
    IN RvSipTransportMgrHandle        hTransportMgr,
    IN RvSipTransportDNSListHandle    hDnsList);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListGetIPElement()

DESCRIPTION

Retrieves the IP address element from the DNS list objects IP addresses list according to the input location.

SYNTAX

```
RvStatus RvSipTransportDNSListGetIPElement(
    IN    RvSipTransportMgrHandle        hTransportMgr,
    IN    RvSipTransportDNSListHandle    hDnsList,
    IN    RvSipListLocation              location,
    INOUT void                           **pRelative,
    OUT   RvSipTransportDNSIPElement     *pIPElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

location

The starting element location.

pRelative

INPUT: The relative host name element for the “next” or “previous” locations.

OUTPUT: The new relative IP element.

pIPElement

The found element.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListPopIPElement()

DESCRIPTION

Retrieves and removes the topmost IP address element from the IP addresses list of the DNS list object.

SYNTAX

```
RvStatus RvSipTransportDNSListPopIPElement(
    IN  RvSipTransportMgrHandle       hTransportMgr,
    IN  RvSipTransportDNSListHandl    hDnsList,
    OUT RvSipTransportDNSIPElement    *pIPElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

pIpElement

The retrieved element.

RETURN VALUES

Returns RvStatus.

RvSipTransportGetNumberOfDNSListEntries()

DESCRIPTION

Retrieves the number of elements in each of the DNS list object lists.

SYNTAX

```
RvStatus RvSipTransportGetNumberOfDNSListEntries(
    IN  RvSipTransportMgrHandle        hTransportMgr,
    IN  RvSipTransportDNSListHandle    hDnsList,
    OUT RvUint32                       *pSrvElements,
    OUT RvUint32                       *pHostNameElements,
    OUT RvUint32                       *pIpAddrElements);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

pSrvElements

The number of SRV elements.

pHostNameElements

The number of host elements.

pIpAddrElements

The number of IP address elements.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListGetUsedHostElement()

DESCRIPTION

Retrieves the host name element that is used to produce the IP list.

SYNTAX

```
RvStatus RvSipTransportDNSListGetUsedHostElement(
    IN  RvSipTransportMgrHandle             pTransportMgr,
    IN  RvSipTransportDNSListHandle         pDnsList,
    OUT RvSipTransportDNSHostNameElement    *pHostElement);
```

PARAMETERS

pTransportMgr

A pointer to the *TransportMgr*.

pDnsList

The handle to the DNS list object.

pHostElement

The host name element that was used last.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListGetUsedSRVElement()

DESCRIPTION

Retrieves the SRV element that is used to produce the host name list.

SYNTAX

```
RvStatus RvSipTransportDNSListGetUsedSRVElement(
    IN  RvSipTransportMgrHandle        pTransportMgr,
    IN  RvSipTransportDNSListHandle    pDnsList,
    OUT RvSipTransportDNSSRVElement    *pSRVElement);
```

PARAMETERS

pTransportMgr

A pointer to the *TransportMgr*.

pDnsList

The handle to the DNS list object.

pSRVElement

The SRV element that was used last.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListSetUsedSRVElement()

DESCRIPTION

Retrieves the SRV element that is used to produce the IP list.

SYNTAX

```
RvStatus RvSipTransportDNSListSetUsedSRVElement(
    IN RvSipTransportMgrHandle        hTransportMgr,
    IN RvSipTransportDNSListHandle    hDnsList,
    IN RvSipTransportDNSSRVElement*   pSRVElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

pHostElement

The host name element structure to be added to the list.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListSetUsedHostElement()

DESCRIPTION

Sets the host name element that is used to produce the IP list.

SYNTAX

```
RvStatus RvSipTransportDNSListSetUsedHostElement(
    IN RvSipTransportMgrHandle              hTransportMgr,
    IN RvSipTransportDNSListHandle          hDnsList,
    IN RvSipTransportDNSHostNameElement*    pHostElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

pHostElement

The host name element structure to be added to the list.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListConstruct()

DESCRIPTION

Allocates and fills the Transport DNS List. The handle to the newly constructed list object is returned by the *phDnsList* parameter. This function also receives the memory pool, page and list pool where the list element and the DNS list object will be allocated.

SYNTAX

```
RvStatus RvSipTransportDNSListConstruct (
  IN  RvSipTransportMgrHandle    hTransportMgr,
  IN  HRPOOL                     hMemPool,
  IN  RvUint32                   maxElementsInSingleDnsList,
  OUT RvSipTransportDNSListHandle
                                 *phDnsList);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hMemPool

The handle to the memory pool

maxElementsInSingleDnsList

The maximum number of elements in a single list.

phDnsList

The handle to the DNS list object.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListDestruct()

DESCRIPTION

Destructs the Transport DNS list that was built by RvSipTransportDNSListConstruct().

SYNTAX

```
RvStatus RvSipTransportDNSListDenstruct(
    IN RvSipTransportMgrHandle        hTransportMgr,
    IN RvSipTransportDNSListHandle    hDnsList);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object to be destructed.

RETURN VALUES

Returns RvStatus.

RvSipTransportDNSListGetSrvElement()

DESCRIPTION

Retrieves an SRV element from the SRV list of the DNS list object.

SYNTAX

```
RvStatus RvSipTransportDNSListGetSrvElement(
    IN    RvSipTransportMgrHandle        hTransportMgr,
    IN    RvSipTransportDNSListHandle    hDnsList,
    IN    RvSipListLocation              location,
    INOUT void                           **pRelative,
    OUT   RvSipTransportDNSSRVElement    *pSrvElement);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hDnsList

The handle to the DNS list object.

location

The location of the element to be retrieved.

pRelative

A pointer to the relative SRV element in the list. This parameter should be provided, if the location is “prev” or “next”. The function will set the value to the retrieved element.

pSrvElement

A pointer to the retrieved SRV element.

RETURN VALUES

Returns RvStatus.

COMPONENTS MODULE

The Components module consists of the Security Server Component module and the Location Database Server Component module.

The Security Server Component module is an example implementation of the SIP Server Security Interface. This implementation includes the DIGEST/MD5 authentication mechanism and a simple User-Password database.

The Location Database Server Component module is an example implementation of a Location Database used to hold the registration information of a user. This implementation uses a hash table for holding all registration records.

This part includes the following section:

6

SERVER COMPONENTS FUNCTIONS

WHAT'S IN THIS SECTION

This section contains the following SIP Server components:

- Security Component Functions
- Location Database Server Component Functions

SECURITY COMPONENT FUNCTIONS

The Security API functions enable implementing the SIP Server Security Interface. This implementation enables the application to manage the User-Password database by adding and removing a user and querying the database for the corresponding password of a specific user. These functions are included in the *RvSipServerSecurityImp.h* header file.

The Security API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvSipServerSecurityMgrImpConstruct()
- RvSipServerSecurityMgrImpDestruct()
- RvSipServerSecurityMgrImpAddUser()
- RvSipServerSecurityMgrImpRemoveUser()
- RvSipServerSecurityMgrImpInitCfg()

RvSipServerSecurityMgrImpConstruct()

DESCRIPTION

Creates and initializes the Security Manager (*SecurityMgr*).

SYNTAX

```
RvSipServerSecurityMgrHandle
RvSipServerSecurityMgrImpConstruct(
    IN RvSipServerMgrHandle      hSipServerMgr,
    IN RvSipServerSecurityCfg    *pCfg,
    IN RvUint32                  size);
```

PARAMETERS

hSipServerMgr

The *SIPServerMgr* handle.

pCfg

The pointer to the *SecurityMgr* configuration structure.

size

The configuration structure size.

RETURN VALUES

If the function succeeded in constructing and initializing the *SecurityMgr*, the return value is a handle to the create object.Otherwise, the return value is NULL.

RvSipServerSecurityMgrImpDestruct()

DESCRIPTION

Destructs the *SecurityMgr* and frees all resources allocated by it.

SYNTAX

```
void RvSipServerSecurityMgrImpDestruct(
    IN RvSipServerSecurityMgrHandle    hSecurityMgr);
```

PARAMETERS

hSecurityMgr

The *SecurityMgr* to destruct.

RETURN VALUES

None.

RvSipServerSecurityMgrImpAddUser()

DESCRIPTION

Adds a user to the *SecurityMgr* users database.

SYNTAX

```
RvStatus RvSipServerSecurityMgrImpAddUser(
    IN RvSipServerSecurityMgrHandle    hSecurityMgr,
    IN const RvChar                    *strUser,
    IN const RvChar                    *strPwd);
```

PARAMETERS

hSecurityMgr

The *SecurityMgr*.

strUser

The user name.

strPwd

The user password.

RETURN VALUES

Returns RvStatus.

RvSipServerSecurityMgrImpRemoveUser()

DESCRIPTION

Removes a user from the Security Manager users database.

SYNTAX

```
RvStatus RvSipServerSecurityMgrImpRemoveUser(
    IN RvSipServerSecurityMgrHandle    hSecurityMgr,
    IN const RvChar                    *strUser);
```

PARAMETERS

hSecurityMgr

The Security Manager.

strUser

The user name to remove.

RETURN VALUES

Returns RvStatus.

RvSipServerSecurityMgrImpInitCfg()

DESCRIPTION

Initializes the configuration parameters specified in the RvSipServerSecurityCfg structure with the Security component default values. You should use this function before calling RvSipServerSecurityMgrImpConstruct() to initialize the configuration structure. Each configuration parameter is initialized with a value of -1, or with a default value. After copying the configuration values received from the application, the -1 values will be replaced with default SIP Server values.

SYNTAX

```
void RvSipServerSecurityMgrImpInitCfg(
    IN    RvInt32                    sizeOfCfg,
    INOUT RvSipServerSecurityCfg*    pSecurityCfg);
```

PARAMETERS

sizeOfCfg

The size of the configuration structure.

pSecurityCfg

IN: A pointer to the security component configuration structure to be initialized.

OUT: The initialized configuration structure.

RETURN VALUES

None.

Get and Set Functions

The Get and Set functions are:

- RvSipServerSecurityImpGetPwd()
- RvSipServerSecurityMgrImpGetResourceStatus()
- RvSipServerSecurityMgrImpSetClientAuthUsername()
- RvSipServerSecurityMgrImpGetClientAuthUsername()
- RvSipServerSecurityMgrImpSetClientAuthPassword()
- RvSipServerSecurityMgrImpGetClientAuthPassword()
- RvSipServerSecurityMgrImpSetNonce()

RvSipServerSecurityImpGetPwd()

DESCRIPTION

Returns the password stored in the Security Manager users database that is related to the specified user. This function receives the user name in a NULL terminated string and returns its password (if found) in the *szPwd* buffer. The *pPwdLen* parameter indicates the password buffer length. If the function succeeds or fails due to RV_InsufficientBuffer, *pPwdLen* will store the actual password length without any ending NULL termination character.

SYNTAX

```
RvStatus RvSipServerSecurityImpGetPwd(
    IN    RvSipServerSecurityMgrHandle    hSecurityMgr,
    IN    const RvChar*                   szUser,
    IN    RvChar*                         szPwd,
    INOUT RvUint32*                       pPwdLen,
    OUT   RvBool*                         pIsFound);
```

PARAMETERS

hSecurityMgr

The Security Manager handle.

szUser

The user name from the authorization header.

szPwd

The returned user password.

pPwdLen

Input: The *szPwd* buffer length.

Output: The returned password length.

pIsFound

Indicates whether such user and its password was found in the Security database.

RETURN VALUES

Returns RvStatus.

RvSipServerSecurityMgrImpGetResourceStatus()

DESCRIPTION

Returns the resource status of the Security Manager.

Note This function is for Debug compilation mode only.

SYNTAX

```
RvStatus RvSipServerSecurityMgrImpGetResourceStatus(
    IN  RvSipServerSecurityMgrHandle     hSecurityMgr,
    OUT RvSipServerSecurityMgrResource   *pResourcesStatus);
```

PARAMETERS

hSecurityMgr

The handle to the Security Manager.

pResourcesStatus

The returned resource status.

RETURN VALUES

Returns RvStatus.

RvSipServerSecurityMgrImpSetClientAuthUsername()

DESCRIPTION

Sets the client-side authentication username.

SYNTAX

```
RvStatus RvSipServerSecurityMgrImpSetClientAuthUsername(
    IN RvSipServerSecurityMgrHandle    hSecurityMgr,
    IN RvChar*                         strUsername);
```

PARAMETERS

hSecurityMgr

The handle to the Security Manager.

strUsername

The username string to be set.

RETURN VALUES

Returns RvStatus.

RvSipServerSecurityMgrImpGetClientAuthUsername()

DESCRIPTION

Returns a copy of the client username. The *pUserLen* parameter indicates the buffer length of *strUsername*. If the function succeeds or fails due to RV_InsufficientBuffer, *pUserLen* will store the actual username length without a NULL termination character.

SYNTAX

```
RvStatus RvSipServerSecurityMgrImpGetClientAuthUsername(
    IN    RvSipServerSecurityMgrHandle    hSecurityMgr,
    IN    RvChar**                        strUsername,
    INOUT RvUint32*                       pUserLen);
```

PARAMETERS

hSecurityMgr

The handle to the Security Manager.

strUsername

The client-side authentication username string.

pUserLen

IN: The buffer size of *strUsername*.

OUT: The returned username length.

RETURN VALUES

Returns RvStatus.

RvSipServerSecurityMgrImpSetClientAuthPassword()

DESCRIPTION

Sets the client-side authentication password.

SYNTAX

```
RvStatus RvSipServerSecurityMgrImpSetClientAuthPassword(
    IN RvSipServerSecurityMgrHandle    hSecurityMgr,
    IN RvChar*                         strPassword);
```

PARAMETERS

hSecurityMgr

The handle to the Security Manager.

strPassword

The password string to be set.

RETURN VALUES

Returns RvStatus.

RvSipServerSecurityMgrImpGetClientAuthPassword()

DESCRIPTION

Returns a copy of the client side authentication password. The *pPwdLen* parameter indicates the buffer length of *strPassword*. If the function succeeds or fails due to RV_InsufficientBuffer, *pPwdLen* will store the actual password length without a NULL termination character.

SYNTAX

```
RvStatus RvSipServerSecurityMgrImpGetClientAuthPassword(
    IN  RvSipServerSecurityMgrHandle    hSecurityMgr,
    IN  RvChar**                        strPassword,
    INOUT RvUint32*                     pPwdLen);
```

PARAMETERS

hSecurityMgr

The handle to the Security Manager.

strPassword

The returned client-side authentication password string.

pPwdLen

IN: The buffer size of *strPassword*.

OUT: The returned password length.

RETURN VALUES

Returns RvStatus.

RvSipServerSecurityMgrImpSetNonce()

DESCRIPTION

Set a new nonce value. The nonce value must be in the following format: <">nonce value<">

SYNTAX

```
RvStatus RvSipServerSecurityMgrImpSetNonce(
    IN RvSipServerSecurityMgrHandle    hSecurityMgr,
    IN const RvChar*                   szNonce);
```

PARAMETERS

hSecurityMgr

The handle to the Security Manager.

szNonce

The new nonce value to set.

RETURN VALUES

Returns RvStatus.

Location Database Server Component Functions

The Location Database API functions enable the application to manage the records in the Location Database. They enable searching for and updating an existing record, inserting a new record, and removing a record. The Location Database API functions also enable constructing and destructing the Location Database module. These functions are included in the *RvProxyLocationDBImp.h* header file.

The Location Database API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvProxyLocationDBImpConstruct()
- RvProxyLocationDBImpDestruct()
- RvProxyLocationDBImpInitCfg()

RvProxyLocationDBImpConstruct()

DESCRIPTION

Creates a *LocationDBMgr* object. Allocates memory and initializes all parameters. Returns a handle to the constructed object.

SYNTAX

```
RvStatus RvProxyLocationDBImpConstruct(
    IN  RvSipServerMgrHandle           hSipServerMgr,
    IN RvProxyLocationDBCfg*           pLocationDBCfg,
    IN RvInt32                         pLocationDBCfgSize,
    OUT RvProxyLocationDBMgrHandle*    hLocationDBMgr);
```

PARAMETERS

hSipServerMgr

The handle to the SIP Server Manager.

pLocationDBCfg

A configuration structure supplied by the application which is used to initialize the Location Database.

pLocationDBCfgSize

The size of the configuration structure.

hLocationDBMgr

A handle to the constructed *LocationDBMgr*.

RETURN VALUES

Returns RvStatus.

RvProxyLocationDBImpDestruct()

DESCRIPTION

Destructs the *LocationDBMgr* and de-allocates all the memory in it.

SYNTAX

```
void RvProxyLocationDBImpDestruct(
    IN RvProxyLocationDBMgrHandle    hLocationDBMgr);
```

PARAMETERS

hLocationDBMgr

The object to be destructed.

RETURN VALUES

None.

RvProxyLocationDBImpInitCfg()

DESCRIPTION

Initiates the Location Database configuration.

SYNTAX

```
void RvProxyLocationDBImpInitCfg(
    IN  RvInt32                  pLDBCfgSize,
    OUT RvProxyLocationDBCfg*    pLDBMgrCfg);
```

PARAMETERS

pLDBCfgSize

The size of the configuration file given as a parameter.

pLDBMgrCfg

The configuration structure to initialize.

RETURN VALUES

None.

Get and Set Functions

The Get and Set functions are:

- RvProxyLocationDBMgrGetResourceStatus()

RvProxyLocationDBMgrGetResourceStatus()

DESCRIPTION

Returns the status of the *LocationDBMgr* resources used.

Note This function is for Debug compilation mode only.

SYNTAX

```
RvProxyLocationDBMgrGetResourceStatus(
    IN  RvProxyLocationDBMgrHandle     hLocationDBMgr,
    OUT RvProxyLocationDBResources*    pResourcesStatus);
```

PARAMETERS

hLocationDBMgr

The handle to *LocationDBMgr*.

pResourcesStatus

The resource structure.

RETURN VALUES

Returns RvStatus.

SIP Server List Module

The SIP Server List module functions of the SIP Server Platform enable you to create and manage a SIP Server List object (*SipServerList*). You can push an element to a *SipServerList* on given location, remove an element from a *SipServerList*, get an element from a given *SipServerList*, or append one *SipServerList* to another.

SipServerList objects are used, for example, to supply the list of contacts when a request is redirected, or to supply the list of Route headers to add to a request that is being forwarded.

This part includes the following section:

7

SIP SERVER LIST FUNCTIONS

WHAT'S IN THIS SECTION

This section contains the SIP Server List functions included in the *RvSipServerList.h* header file.

The SIP Server List API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvSipServerListConstruct()
- RvSipServerListDestruct()
- RvSipServerListPushElem()
- RvSipServerListRemoveElem()
- RvSipServerListIsEmpty()
- RvSipServerListConstructElemInList()
- RvSipServerListAppend()
- RvSipServerStdListAllocElem()
- RvSipServerStdListConstructFromCfg

RvSipServerListConstruct()

DESCRIPTION

Constructs a *SipServerList*.

SYNTAX

```
RvStatus RvSipServerListConstruct(
    IN  RvSipServerListPoolHandle    hListPool,
    OUT RvSipServerListHandle        *hList);
```

PARAMETERS

hListPool

The handle to the List Pool.

hList

A handle to the created list.

RETURN VALUES

Returns RvStatus.

RvSipServerListDestruct()

DESCRIPTION

Destructs a *SipServerList*. When the list is destructed, it frees all stored data (header and messages).

SYNTAX

```
RvStatus RvSipServerListDestruct(
    IN RvSipServerListHandle    hList);
```

PARAMETERS

hHeaderList

The handle to the list to destruct.

RETURN VALUES

Returns RvStatus.

RvSipServerListPushElem()

DESCRIPTION

Adds a given element to the *SipServerList*.

SYNTAX

```
RvStatus RvSipServerListPushElem(
    IN    RvSipServerListHandle        hList,
    IN    void                         *pData,
    IN    RvSipServerListElemType      eType,
    IN    RvSipServerElemLocation      eLocation,
    IN    RvSipServerListElemHandle    hPos,
    INOUT RvSipServerListElemHandle    *pNewPos);
```

PARAMETERS

hList

The handle to the list.

pData

A pointer to the element data to be added to the list.

eType

The element type.

eLocation

The inserted element location, such as first and last.

hPos

The current list position, relevant in the case where the location is NEXT or PREV.

pNewPos

The returned location of the object that was pushed.

RETURN VALUES

Returns RvStatus.

RvSipServerListRemoveElem()

DESCRIPTION

Removes an element from the *SipServerList*.

SYNTAX

```
RvStatus RvSipServerListRemoveElem(
    IN RvSipServerListHandle        hList,
    IN RvSipServerListElemHandle    hListElem);
```

PARAMETERS

hList

The handle to the list.

hListElem

The handle to the list element to be removed.

RvSipServerListIsEmpty()

DESCRIPTION

Indicates whether or not the *SipServerList* is empty.

SYNTAX

```
RvStatus RvSipServerListIsEmpty(
    IN  RvSipServerListHandle    hList,
    OUT RvBool                   *pbIsEmpty);
```

PARAMETERS

hList

The handle to the list.

pbIsEmpty

RV_TRUE if the *SipServerList* is empty. Otherwise RV_FALSE.

RETURN VALUES

Returns RvStatus.

RvSipServerListConstructElemInList()

DESCRIPTION

Creates a new element. This function adds the new element to the *SipServerList* and returns a pointer to the newly created element so the application can set its data.

SYNTAX

```
RvStatus RvSipServerListConstructElemInList(
    IN  RvSipServerListHandle        hList,
    IN  RvSipServerListElemType      eType,
    IN  RvSipAddressType             eAddrType,
    IN  RvSipServerElemLocation      eListLocation,
    IN  RvSipServerListElemHandle    hPos,
    OUT void                         **pData,
    OUT RvSipServerListElemHandle    *pNewPos);
```

PARAMETERS

hList

The handle to the list.

eType

The element type.

eAddrType

The address type. This parameter will be used only if *eType* is RVSIPSERVER_ELEM_TYPE_ADDRESS.

eListLocation

The location in the list where the element should be added, such as first or last.

hPos

The current list element, used only if *eListLocation* is NEXT or PREV.

pData

The returned address to the element data which currently has been added to the list.

pNewPos

The returned list element.

Return Values

Returns RvStatus.

RvSipServerListAppend()

DESCRIPTION

Appends the content of the *hSrcList* list to the *hTrgList* list. If *hSrcList* is empty, all the list elements of *hTrgList* will be de-allocated.

SYNTAX

```
RvStatus RvSipServerListAppend(
    IN RvSipServerListHandle    hSrcList,
    IN RvSipServerListHandle    hTrgList);
```

PARAMETERS

hSrcList

The source list.

hTrgList

The target list.

RETURN VALUES

Returns RvStatus.

RvSipServerStdListAllocElem()

DESCRIPTION

Allocates a list element using the internal list allocator. After the list element is allocated, RvSipServerStdListAddElem() should be used to add the element to the list. This function allows specifying data size optionally in order to allocate data space using the element allocator. The key handle value for newly allocated elements is not defined. The data handle value may be defined if dataSize > 0 as described in the *dataSize* parameter description.

SYNTAX

```
RvSipServerStdListElemHandle RvSipServerStdListAllocElem(
    IN  RvSipServerStdListHandle    hList,
    IN  RvSize_t                    dataSize);
```

PARAMETERS

hList

The handle of the list object.

dataSize

The size of data to be allocated. If the value is > 0, additional storage space will be allocated just after the element object instance, and the data handle will contain a pointer to this area. Otherwise, data value is undefined.

RETURN VALUES

Returns RvSipServerStdListElemHandle (handle of the allocated element).

RvSipServerStdListConstructFromCfg

DESCRIPTION

Constructs a list object using the specified list configuration

SYNTAX

```
RvStatus RVCALLCONV RvSipServerStdListConstructFromCfg(
    IN  RvSipServerStdListInstance*    pInstList,
    IN  RvSipServerStdListElemIntf*    pElemInf,
    IN  RvSipServerStdListCfg*         pCfg,
    OUT RvSipServerStdListHandle*     phList);
```

PARAMETERS

hInstList

The handle of a list instance to construct.

pElemInf

The address of an element interface record. please refer to RvSipServerStdListElemIntf for a full description of members. Function pointers passed with NULL value will have the following default values:

- pElemInf.pfnCompare—equal operator comparison.
- pElemInf.pfnAlloc—process heap allocation.
- pElemInf.pfnFree—process heap deallocation.

pCfg

A pointer to a list configuration structure.

phList

The handle of constructed list.

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

- RvSipServerListGetHead()
- RvSipServerListGetTail()
- RvSipServerListGetNext()
- RvSipServerListGetPrev()
- RvSipServerListElemGetData()
- RvSipServerListElemGetDataType()

RvSipServerListGetHead()

DESCRIPTION

Returns the first element from the *SipServerList*.

SYNTAX

```
RvStatus RvSipServerListGetHead(
    IN  RvSipServerListHandle        hList,
    OUT RvSipServerListElemHandle    *phListElem);
```

PARAMETERS

hList

The handle to the list.

phListElem

The returned element.

RETURN VALUES

Returns RvStatus.

RvSipServerListGetTail()

DESCRIPTION

Returns the last element from the *SipServerList*.

SYNTAX

```
RvStatus RvSipServerListGetTail(
    IN  RvSipServerListHandle        hList,
    OUT RvSipServerListElemHandle    *phElem);
```

PARAMETERS

hList

The handle to the list.

phElem

The returned element.

RETURN VALUES

Returns RvStatus.

RvSipServerListGetNext()

DESCRIPTION

Returns the next element from the *SipServerList*.

SYNTAX

```
RvStatus RvSipServerListGetNext(
    IN  RvSipServerListHandle        hList,
    IN  RvSipServerListElemHandle    hPos,
    OUT RvSipServerListElemHandle    *pNewPos);
```

PARAMETERS

hList

The handle to the list.

hPos

The current list element.

pNewPos

The returned element.

RETURN VALUES

Returns RvStatus.

RvSipServerListGetPrev()

DESCRIPTION

Returns the previous element form the *SipServerList*.

SYNTAX

```
RvStatus RvSipServerListGetPrev(
    IN  RvSipServerListHandle        hList,
    IN  RvSipServerListElemHandle    hPos,
    OUT RvSipServerListElemHandle    *pNewPos);
```

PARAMETERS

hList

The handle to the list.

hPos

The current list element.

pNewPos

The returned element.

RETURN VALUES

Returns RvStatus.

RvSipServerListElemGetData()

DESCRIPTION

Returns the object stored in the *SipServerList* element.

SYNTAX

```
void *RvSipServerListElemGetData(
    IN RvSipServerListElemHandle    listElem);
```

PARAMETERS

listElem

The list element.

RETURN VALUES

Returns the handle to the stored element data.

RvSipServerListElemGetDataType()

DESCRIPTION

Returns the type of the stored element.

SYNTAX

```
RvSipServerListElemType RvSipServerListElemGetDataType(
    IN RvSipServerListElemHandle    listElem);
```

PARAMETERS

listElem

The list element.

RETURN VALUES

Returns the list element type.

SIP Server STD Layer

The SIP Server STD layer contains a set of non-SIP-related general purpose utilities such as the XML library, generic list, and allocator interface for use by the SIP Server and the application.

This part includes the following section:

- SIP Server STD Functions

8

SIP SERVER STD FUNCTIONS

WHAT'S IN THIS SECTION

This section contains the SIP Server STD API functions included in the *RvSipServerStdList.h* and *RvSipServerStdBase.h* header files, and the XML header files for each XML object (*XMLMgr*, *XMLTag*, *XMLDoc*, *XMLAttr,* *XPath,* and so on).

The SIP Server STD API includes:

- STD List Functions
- STD List Iterator Functions
- STD List Element Functions
- STD Allocator Functions
- XML Manager Functions
- XML Tag Functions
- XML Document Functions
- XML Attribute Functions
- XML Xpath Functions

STD List Functions

This section includes:

CONTROL FUNCTIONS

This Control functions are:

- RvSipServerStdListElemKeyCompareFunc()
- RvSipServerStdListElemCopyFunc()
- RvSipServerStdListConstruct()
- RvSipServerStdListDestruct()
- RvSipServerStdListLock()
- RvSipServerStdListUnlock()
- RvSipServerStdListFindElem()
- RvSipServerStdListFind()
- RvSipServerStdListAddHead()
- RvSipServerStdListAddTail()
- RvSipServerStdListAddElem()
- RvSipServerStdListAllocElem()
- RvSipServerStdListRemoveElem()
- RvSipServerStdListRemove()
- RvSipServerStdListClear()
- RvSipServerStdListIsEmpty()
- RvSipServerStdListFindSourceExclusive()
- RvSipServerStdListCopyList()

RvSipServerStdListElemKeyCompareFunc()

DESCRIPTION

Compares the keys of two list elements (*ListElems*).

SYNTAX

```
typedef RvBool (RVCALLCONV
*RvSipServerStdListElemKeyCompareFunc)(
    IN RvSipServerStdListHandle           hList,
    IN RvSipServerStdListElemKeyHandle    hKey1,
    IN RvSipServerStdListElemKeyHandle    hKey2);
```

PARAMETERS

hList

The handle of the *ListElem*.

hKey1

The handle of key 1.

hKey2

The handle of key 2.

RETURN VALUES

Returns RV_TRUE if the *ListElem* objects match. Otherwise, returns RV_FALSE.

RvSipServerStdListElemCopyFunc()

DESCRIPTION

Copies a source *ListElem* into a destination *ListElem*.

SYNTAX

```
RvStatus RvSipServerStdListElemCopyFunc(
    IN RvSipServerStdListHandle        hList,
    IN RvSipServerStdListElemHandle    hSrcElem,
    IN RvSipServerStdListElemHandle    hDstElem);
```

PARAMETERS

hList

The handle of the *ListElem*.

hSrcElem

The handle of the source *ListElem*.

hDstElem

The handle of the destination *ListElem*.

RETURN VALUES

Returns RvStatus.

RvSipServerStdListConstruct()

DESCRIPTION

Constructs a *ListElem* which is thread-safe.

SYNTAX

```
RvStatus RvSipServerStdListConstruct(
    IN  RvSipServerStdListInstance*   hInstList,
    IN  RvSipServerLogHandle          hLog,
    IN  RvSipServerStdListElemIntf*   pElemInf,
    OUT RvSipServerStdListHandle*     phList);
```

PARAMETERS

hInstList

The handle of the *ListElem* instance to construct.

hLog

The handle of the log object.

pElemInf

The address of an element interface record. See RvSipServerStdListElemIntf for a full description of the members. The function pointers passed with a NULL value will have the following default values:

- pElemInf.pfnCompare—equal operator comparison
- pElemInf.pfnAlloc—process heap allocation
- pElemInf.pfnFree—process heap de-allocation

phList

The handle of the constructed *ListElem*.

RETURN VALUES

Returns RvStatus.

RvSipServerStdListDestruct()

DESCRIPTION

Destructs a *ListElem*.

SYNTAX

```
void RvSipServerStdListDestruct(
    IN RvSipServerStdListHandle    hList);
```

PARAMETERS

hList

The handle of the *ListElem*.

RETURN VALUES

None.

RvSipServerStdListLock()

DESCRIPTION

Locks the *ListElem*.

SYNTAX

```
void RvSipServerStdListLock(
    IN RvSipServerStdListHandle    hList);
```

PARAMETERS

hList

The handle of the *ListElem*.

RETURN VALUES

None.

RvSipServerStdListUnlock()

DESCRIPTION

Unlocks the *ListElem*.

SYNTAX

```
void RvSipServerStdListUnlock(
    IN RvSipServerStdListHandle    hList);
```

PARAMETERS

hList

The handle of the *ListElem*.

RETURN VALUES

None.

RvSipServerStdListFindElem()

DESCRIPTION

Finds and returns an element of the *ListElem*, given a key and, optionally, a custom key compare function. If a *ListElem* is not found, the last *ListElem* of the list is returned, or NULL if list is empty.

SYNTAX

```
RvBool RvSipServerStdListFindElem(
    IN  RvSipServerStdListHandle               hList,
    IN  RvSipServerStdListElemKeyHandle        hKey,
    IN  RvSipServerStdListElemKeyCompareFunc   pfnCompare,
    OUT RvSipServerStdListElemHandle*          phElem);
```

PARAMETERS

hList

The handle of the *ListElem*.

hKey

The key of the *ListElem* to find.

pfnCompare

The custom key compare function for this find operation. If NULL, the default compare function will be used.

phElem

The handle of the found *ListElem*.

RETURN VALUES

Returns RV_TRUE if the *ListElem* is found. The *phElem* parameter holds the element address. Returns RV_FALSE if the *ListElem* is not found. The *phElem* parameter holds the tail list *ListElem*, or NULL if the list is empty.

RvSipServerStdListFind()

DESCRIPTION

Finds and returns the data of a *ListElem* that was given an *ListElem* key and, optionally, a custom key compare function.

SYNTAX

```
RvBool RvSipServerStdListFind(
    IN RvSipServerStdListHandle                hList,
    IN RvSipServerStdListElemKeyHandle         hKey,
    IN RvSipServerStdListElemKeyCompareFunc    pfnCompare,
    OUT RvSipServerStdListElemDataHandle*      pData);
```

PARAMETERS

hList

The handle of the *ListElem*.

hKey

The key of the *ListElem* to find.

pfnCompare

The custom key compare function for this find operation. If NULL, the default compare function will be used.

pData

The data of the found *ListElem*.

RETURN VALUES

Returns RV_TRUE if the *ListElem* is found. The *pData* parameter holds the *ListElem* data. Returns RV_FALSE if the *ListElem* is not found. The *pData* parameter value is undefined.

RvSipServerStdListAddHead()

DESCRIPTION

Adds a *ListElem* to the head of a list.

SYNTAX

```
RvStatus RvSipServerStdListAddHead(
    IN RvSipServerStdListHandle           hList,
    IN RvSipServerStdListElemKeyHandle    hKey,
    IN RvSipServerStdListElemDataHandle   hData);
```

PARAMETERS

hList

The handle of the *ListElem*.

hKey

The added *ListElem* key handle.

hData

The added *ListElem* data handle.

RETURN VALUES

Returns RvStatus.

RvSipServerStdListAddTail()

DESCRIPTION

Adds a *ListElem* to the tail of a list, or after a specific *ListElem*.

SYNTAX

```
RvStatus RvSipServerStdListAddTail(
    IN RvSipServerStdListHandle            hList,
    IN RvSipServerStdListElemKeyHandle     hKey,
    IN RvSipServerStdListElemDataHandle    hData,
    IN RvSipServerStdListElemHandle        hPrevElem);
```

PARAMETERS

hList

The handle of the *ListElem*.

hKey

The added *ListElem* key handle.

hData

The added *ListElem* data handle.

hPrevElem

The *ListElem* after which the new *ListElem* will be added. If NULL, the new *ListElem* will be added at the end of the list.

RETURN VALUES

Returns RvStatus.

RvSipServerStdListAddElem()

DESCRIPTION

Adds an *ListElem* to the list. After the *ListElem* is added, it cannot be added to an additional list. To move a *ListElem* from one list to another, the *ListElem* must first be removed from the first list and then added to the second. A *ListElem* can be created using RvSipServerStdListAllocElem().

Note A *ListElem* can be added only to lists that have the allocator that was used to allocate the *ListElem*.

SYNTAX

```
RvStatus RvSipServerStdListAddElem(
    IN RvSipServerStdListHandle        hList,
    IN RvSipServerStdListElemHandle    hElem);
```

PARAMETERS

hList

The handle of the *ListElem*.

hElem

The added *ListElem* handle.

RETURN VALUES

Returns RvStatus.

RvSipServerStdListAllocElem()

DESCRIPTION

Allocates a *ListElem* using the internal list allocator (*ListAllocator*). After the *ListElem* was allocated, RvSipServerStdListAddElem() should be used to add the *ListElem* to the list. This function allows specifying the data size optionally in order to allocate data space using the *ListAllocator* of the *ListElem.* The key handle value for the newly allocated *ListElem*s is not defined. The data handle value may be defined if the *dataSize* is greater than 0, as described in the dataSize parameter description.

SYNTAX

```
RvSipServerStdListElemHandle RvSipServerStdListAllocElem(
    IN RvSipServerStdListHandle    hList,
    IN RvSize_t                    dataSize);
```

PARAMETERS

hList

The handle of the *ListElem*.

dataSize

The size of data to be allocated. If the value is greater than 0, additional storage space will be allocated just after the *ListElem* instance, and the data handle will contain a pointer to this area. Otherwise, the data value is undefined.

RETURN VALUES

Returns the *ListElem*, or NULL if the function failed to allocate the *ListElem*.

RvSipServerStdListRemoveElem()

DESCRIPTION

Removes and de-allocates a *ListElem* that was given a *ListElem* handle.

SYNTAX

```
RvStatus RvSipServerStdListRemoveElem(
    IN  RvSipServerStdListHandle        hList,
    IN  RvSipServerStdListElemHandle    hElem,
    OUT RvBool*                         pbListIsEmpty);
```

PARAMETERS

hList

The handle of the *ListElem*.

hElem

The handle of the *ListElem* to be removed. This handle becomes invalid if the function succeeds.

pbListIsEmpty

This parameter is optional.

Indicates if the list became empty after *ListElem* removal. NULL is a valid value for this parameter.

RETURN VALUES

Returns RvStatus.

RvSipServerStdListRemove()

DESCRIPTION

Removes and de-allocates a *ListElem* that was given a *ListElem* key handle.

SYNTAX

```
RvStatus RvSipServerStdListRemove(
    IN  RvSipServerStdListHandle          hList,
    IN  RvSipServerStdListElemKeyHandle   hKey,
    OUT RvBool*                           pbListIsEmpty);
```

PARAMETERS

hList

The handle of the *ListElem*.

hKey

The handle of the *ListElem* key to be removed.

pbListIsEmpty

This parameter is optional.

Indicates if the list became empty after the *ListElem* was removal. NULL is a valid value for this parameter.

RETURN VALUES

Returns RvStatus.

RvSipServerStdListClear()

DESCRIPTION

Clears the list of all *ListElem* objects, de-allocating *ListElem* memory.

SYNTAX

```
RvStatus RvSipServerStdListClear(
    IN RvSipServerStdListHandle    hList);
```

PARAMETERS

hList

The handle of the *ListElem*.

RETURN VALUES

Returns RvStatus.

RvSipServerStdListIsEmpty()

DESCRIPTION

Checks if the list is empty.

SYNTAX

```
RvBool RvSipServerStdListIsEmpty(
    IN RvSipServerStdListHandle    hList);
```

PARAMETERS

hList

The handle of the *ListElem*.

RETURN VALUES

Returns RV_TRUE if the list is empty. Otherwise, returns RV_FALSE.

RvSipServerStdListFindSourceExclusive()

DESCRIPTION

Copies all the *ListElem* objects of the source list (which do not exist in the compared list) to the result list.

SYNTAX

```
RvStatus RvSipServerStdListFindSourceExclusive(
    IN  RvSipServerStdListHandle          hSrcList,
    IN  RvSipServerStdListHandle          hCompareList,
    IN  RvSipServerStdListElemCopyFunc    pfnCpy,
    OUT RvSipServerStdListHandle          hResultList);
```

PARAMETERS

hScrList

The handle of the source *ListElem*.

hCompareList

The handle of the compared *ListElem*.

pfnCpy

The custom *ListElem* copy function.

hResultList

The handle of the result *ListElem*.

RETURN VALUES

Returns RvStatus.

RvSipServerStdListCopyList()

DESCRIPTION

Copies all the *ListElem* objects of the source list to the destination list.

SYNTAX

```
RvStatus RvSipServerStdListCopyList(
    IN  RvSipServerStdListHandle          hSrcList,
    IN  RvSipServerStdListElemCopyFunc    pfnCpy,
    OUT RvSipServerStdListHandle          hDstList);
```

PARAMETERS

hSrcList

The handle of the source *ListElem*.

pfnCpy

The custom *ListElem* copy function.

hDstList

The handle of the destination *ListElem*.

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

The Get function is:

- RvSipServerStdListGetLength()

RvSipServerStdListGetLength()

DESCRIPTION

Gets number of *ListElem* objects in the list.

SYNTAX

```
RvUint16 RvSipServerStdListGetLength(
    IN RvSipServerStdListHandle    hList);
```

PARAMETERS

hList

The handle of the *ListElem*.

RETURN VALUES

Returns the number of *ListElem* objects in the list.

STD List Iterator Functions

This section includes:

Get and Set Functions

The Get functions are:

- RvSipServerStdListIteratorGetFirst()
- RvSipServerStdListIteratorGetNext()

RvSipServerStdListIteratorGetFirst()

DESCRIPTION

Instantiates an internal iterator and gets the first *ListElem*.

Note This function does not lock the list. Developers should lock the list before calling this function and throughout the iteration if required.

SYNTAX

```
RvSipServerStdListElemHandle
RvSipServerStdListIteratorGetFirst(
    IN RvSipServerStdListHandle    hList);
```

PARAMETERS

hList

The handle of the *ListElem*.

RETURN VALUES

Returns the first *ListElem* of RvSipServerListElemHandleDescription, or NULL if the list is empty.

RvSipServerStdListIteratorGetNext()

DESCRIPTION

Gets the next *ListElem* as pointed to by the internal iterator which was created by RvSipServerStdListIteratorGetFirst().

Note This function does not lock the list. Developers should lock the list before calling this function and throughout the iteration if required.

SYNTAX

```
RvSipServerStdListElemHandle
RvSipServerStdListIteratorGetNext(
    IN RvSipServerStdListHandle    hList);
```

PARAMETERS

hList

The handle the *ListElem*.

RETURN VALUES

Returns the next *ListElem* of RvSipServerStdListElemHandle, or NULL if the list is empty.

STD List Element Functions

This section includes:

CONTROL FUNCTIONS

The Control functions are:

- RvSipServerStdListElemKeyCompareFuncStrings()
- RvSipServerStdListElemCopyFuncShallow()

RvSipServerStdListElemKeyCompareFuncStrings()

DESCRIPTION

An implementation of RvSipServerStdListElemKeyCompareFunc() that compares two NULL terminated string keys.

SYNTAX

```
RvBool RvSipServerStdListElemKeyCompareFuncStrings(
    IN RvSipServerStdListHandle          hList,
    IN RvSipServerStdListElemKeyHandle   hKey1,
    IN RvSipServerStdListElemKeyHandle   hKey2);
```

PARAMETERS

hList

The handle of the *ListElem*.

hKey1

The handle of key 1.

hKey2

The handle of key 2.

RETURN VALUES

Returns RV_TRUE if the *ListElem* objects match. Otherwise, returns RV_FALSE.

RvSipServerStdListElemCopyFuncShallow()

DESCRIPTION

An implementation of RvSipServerStdListElemCopyFunc() that performs a shallow copy of the source *ListElem* key and data into the destination *ListElem*. A shallow copy is a trivial assignment of a source *ListElem* value into the destination *ListElem*.

SYNTAX

```
RvStatus RvSipServerStdListElemCopyFuncShallow(
    IN RvSipServerStdListHandle        hList,
    IN RvSipServerStdListElemHandle    hSrcElem,
    IN RvSipServerStdListElemHandle    hDstElem);
```

PARAMETERS

hList

The handle of the *ListElem*.

hSrcElem

The handle of the source *ListElem*.

hDstElem

The handle of the destination *ListElem*.

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

The Get and Set functions are:

- RvSipServerStdListElemGetKey()
- RvSipServerStdListElemGetData()
- RvSipServerStdListElemSetKey()
- RvSipServerStdListElemSetData()

RvSipServerStdListElemGetKey()

DESCRIPTION

Gets the key handle of a *ListElem*.

SYNTAX

```
RvSipServerStdListElemKeyHandle
RvSipServerStdListElemGetKey(
    IN RvSipServerStdListElemHandle    hElem);
```

PARAMETERS

hElem

The handle of the *ListElem*.

RETURN VALUES

Returns the *ListElem* key handle.

RvSipServerStdListElemGetData()

DESCRIPTION

Gets the data handle of a *ListElem*.

SYNTAX

```
RvSipServerStdListElemDataHandle
RvSipServerStdListElemGetData(
    IN RvSipServerStdListElemHandle    hElem);
```

PARAMETERS

hElem

The handle of the *ListElem*.

RETURN VALUES

Returns the *ListElem* data handle.

RvSipServerStdListElemSetKey()

DESCRIPTION

Sets the key handle of a *ListElem*.

SYNTAX

```
void RvSipServerStdListElemSetKey(
    IN RvSipServerStdListElemHandle       hElem,
    IN RvSipServerStdListElemKeyHandle    hKey);
```

PARAMETERS

hElem

The handle of the *ListElem*.

hKey

The key handle.

RETURN VALUES

None.

RvSipServerStdListElemSetData()

DESCRIPTION

Sets the data handle of a *ListElem*.

SYNTAX

```
void RvSipServerStdListElemSetData(
    IN RvSipServerStdListElemHandle        hElem,
    IN RvSipServerStdListElemDataHandle    hData);
```

PARAMETERS

hElem

The handle of the *ListElem* object.

hData

The data handle.

RETURN VALUES

None.

STD Allocator Functions

This section includes:

CONTROL FUNCTIONS

The Control functions are:

- RvSipServerStdAllocatorAllocFunc()
- RvSipServerStdAllocatorFreeFunc()

RvSipServerStdAllocatorAllocFunc()

DESCRIPTION

Allocates a memory buffer with a specified size.

SYNTAX

```
typedef void* (RVCALLCONV
*RvSipServerStdAllocatorAllocFunc)(
    IN RvSipServerStdAllocatorHandle    hAllocator,
    IN RvSize_t                         size;
```

PARAMETERS

hAllocator

The handle of the Allocator.

size

The size of the buffer to allocate.

RETURN VALUES

Returns the address of the allocated buffer, or NULL if allocation failed.

RvSipServerStdAllocatorFreeFunc()

DESCRIPTION

Frees a memory buffer with a specified size. The buffer should have been allocated using theRvSipServerStdAllocatorAllocFunc() of the given allocator (*Allocator*).

SYNTAX

```
typedef void (RVCALLCONVRvSipServerStdAllocatorFreeFunc)(
    IN RvSipServerStdAllocatorHandle    hAllocator,
    IN void*                            pv);
```

PARAMETERS

hAllocator

The handle of an *Allocator*.

pv

The address of the buffer to free.

RETURN VALUES

None.

XML MANAGER FUNCTIONS

The XML Manager (*XMLMgr*) manages the collection of all the XML objects. This section contains the XML Manager API functions found in the *RvXMLMgr.h* header file.

The XML Manager API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvXMLMgrConstruct()
- RvXMLMgrDestruct()

RvXMLMgrConstruct()

DESCRIPTION

Constructs an *XMLMgr*.

SYNTAX

```
RvStatus RvXMLMgrConstruct(
    IN  RvXMLMgrCfg       *pXMLMgrCfg,
    OUT RvXMLMgrHandle    *phXMLMgr);
```

PARAMETERS

pXMLMgrCfg

The configuration structure with which to initialize the *XMLMgr*.

phXMLMgr

A pointer to the *XMLMgr* handle, which will contain a handle for a valid *XMLMgr* instance in case of a successful call to this function.

RETURN VALUES

Returns RvStatus.

RvXMLMgrDestruct()

DESCRIPTION

Destroys an *XMLMgr*.

SYNTAX

```
RvStatus RvXMLMgrDestruct(
    IN RvXMLMgrHandle    hXMLMgr);
```

PARAMETERS

hXMLMgr

The *XMLMgr* handle to the instance to be destructed. The handle becomes invalid after calling this function.

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

The Get and Set functions are:

- RvXMLMgrGetAppContext()
- RvXMLMgrGetResourceStatus()

RvXMLMgrGetAppContext()

DESCRIPTION

Gets the handle of the application context.

SYNTAX

```
RvStatus RvXMLMgrGetAppContext(
    IN  RvXMLMgrHandle    hXMLMgr,
    OUT void**            phAppContext);
```

PARAMETERS

hXMLMgr

The *XMLMgr* handle.

phAppContext

The handle to the application context that was assigned to the XML builder configuration.

RETURN VALUES

Returns RvStatus.

RvXMLMgrGetResourceStatus()

DESCRIPTION

Returns the status of the *XMLMgr* resources that were used.

Note This function is for use in debug mode only.

SYNTAX

```
RvStatus RvXMLMgrGetResourceStatus(
    IN  RvXMLMgrHandle    hXMLMgr,
    OUT RvXMLResources*   pResourcesStatus);
```

PARAMETERS

hXMLMgr

The handle to the *XMLMgr*.

pResourcesStatus

The structure of the resources.

RETURN VALUES

Returns RvStatus.

XML TAG FUNCTIONS

An XML Tag object (*XMLTag*) is an element of an XML document (*XMLDoc*).An *XMLTag* is a recursive object that holds a list of child *XMLTag* objects. The *XMLTag* also holds a list of attributes (*XMLAttr*). The XML Tag API functions allow you to create and destruct *XMLTag* objects, and to create remark tags. This section contains the XML Tag API functions found in the *RvXMLTag.h* header file.

The XML Tag API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvXMLTagCreate()
- RvXMLTagDestruct()
- RvXMLTagRemarkCreate()

RvXMLTagCreate()

DESCRIPTION

Creates a new *XMLTag* in an *XMLDoc*. The *XMLTag* will be added to the parent *XMLTag* list according to the *eLocation* parameter (first/last/next/prev). The *hCurrTag* parameter will be used as reference when the location is next/prev. The name and data can be supplied and will be used for encoding in the following format:

```
<strName>strData<\strName>
```

SYNTAX

```
RvStatus RvXMLTagCreate(
    IN  RvXMLDocHandle       hDoc,
    IN  RvXMLTagHandle       hParent,
    IN  RvXMLTagHandle       hCurrTag,
    IN  RvXMLListLocation    eLocation,
    IN  RvChar*              strName,
    IN  RvChar*              strData,
    OUT RvXMLTagHandle*      phTag);
```

PARAMETERS

hDoc

The handle to the *XMLDoc* in which to create the new *XMLTag*.

hParent

The *XMLTag* which is the parent of the new *XMLTag*. The *XMLTag* will be added to the *XMLTag* list of the parent.

hCurrTag

The current *XMLTag* to which the new *XMLTag* will be added (when the location is next/prev).

eLocation

The location of the new *XMLTag* (first/last/next/prev).

strName

The name of the new *XMLTag*.

StrData

The data of the new *XMLTag*.

phTag

A handle to the new *XMLTag* that was created.

Return Values

Returns RvStatus.

RvXMLTagDestruct()

DESCRIPTION

Destructs an *XMLTag*. The *XMLTag* will be removed from the *XMLDoc* and all the child *XMLTag* objects of the *XMLTag* will be removed recursively.

SYNTAX

```
RvStatus RvXMLTagDestruct(
    IN RvXMLTagHandle    hTag);
```

PARAMETERS

hTag

The handle to the *XMLTag* to destruct.

RETURN VALUES

Returns RvStatus.

RvXMLTagRemarkCreate()

DESCRIPTION

Creates a new *XMLTag* in an *XMLDoc*. The *XMLTag* will be added to the parent *XMLTag* list according to the eLocation parameter (first/last/next/prev). The *hCurrTag* parameter will be used as reference when the location is next/prev.

The data can be supplied and will be used for encoding in the following format:

```
<!-- strRemark -->
```

SYNTAX

```
RvStatus RvXMLTagRemarkCreate(
    IN  RvXMLDocHandle       hDoc,
    IN  RvXMLTagHandle       hParent,
    IN  RvXMLTagHandle       hCurrTag,
    IN  RvXMLListLocation    eLocation,
    IN  RvChar*              strRemark,
    OUT RvXMLTagHandle*      phTag);
```

PARAMETERS

hDoc

The handle to the *XMLDoc* in which to create the new XML.

hParent

The *XMLTag* which is the parent of the new *XMLTag*. The *XMLTag* will be added to the *XMLTag* list of the parent.

hCurrTag

The current *XMLTag* to which the new *XMLTag* will be added (when the location is next/prev).

eLocation

The location of the new *XMLTag* (first/last/next/prev).

strRemark

The remark string.

phTag

The handle to the new *XMLTag* that was created.

Return Values

Returns RvStatus.

GET AND SET FUNCTIONS

The Get and Set functions are:

- RvXMLTagGetChildTag()
- RvXMLTagGetChildTagByName()
- RvXMLTagGetParentTag()
- RvXMLTagSetName()
- RvXMLTagGetName()
- RvXMLTagSetData()
- RvXMLTagGetData()
- RvXMLTagGetAttribute()
- RvXMLTagGetAttributeByName()
- RvXMLTagGetDoc()
- RvXMLTagCompare()
- RvXMLTagFindTuple()
- RvXMLTagCopy()
- RvXMLTagGetChildTagByNameEx()
- RvXMLTagSetNameSpace()
- RvXMLTagGetNameSpace()
- RvXMLTagGetAttributeByNameEx()
- RvXMLTagFindTupleEx()
- RvXMLTagSetType()
- RvXMLTagGetType()
- RvXMLTagNSLocalToGlobal()
- RvXMLGetNamespaceAttr()
- RvXMLGetNamespaceAttrEx()
- RvXMLGetDefaultNamespaceAttr()
- RvXMLCreateNamespaceAttr()
- RvXMLTagNameCompareWns()
- RvXMLTagNextDescendant()
- RvXMLTagEncode()

RvXMLTagGetChildTag()

DESCRIPTION

Returns a child *XMLTag* according to the location argument (first/last/next/prev). The *hCurrTag* is the current tag from which to get the *XMLTag* if the location is next/prev.

SYNTAX

```
RvStatus RvXMLTagGetChildTag(
    IN  RvXMLTagHandle       hParent,
    IN  RvXMLTagHandle       hCurrTag,
    IN  RvXMLListLocation    eLocation,
    OUT RvXMLTagHandle*      phTag);
```

PARAMETERS

hParent

The parent *XMLTag* which the *XMLTag* should be returned from its list.

hCurrTag

The current *XMLTag* to which the new *XMLTag* will be added (when the location is next/prev).

eLocation

The *XMLTag* location (first/last/prev/next).

phTag

The *XMLTag* that was returned.

RETURN VALUES

Returns RvStatus.

RvXMLTagGetChildTagByName()

DESCRIPTION

Returns a child *XMLTag* for a given *XMLTag* by the name of the tag. If NULL is returned, no child tag with the given name exists for this tag.

SYNTAX

```
RvStatus RvXMLTagGetChildTagByName(
    IN  RvXMLTagHandle     hParent,
    IN  RvChar*            strName,
    OUT RvXMLTagHandle*    phTag);
```

PARAMETERS

hParent

The parent *XMLTag* that contains the *XMLTag* that should be returned from its list.

strName

The name of the tag which should be returned.

phTag

The *XMLTag* that was returned.

RETURN VALUES

Returns RvStatus.

RvXMLTagGetParentTag()

DESCRIPTION

Returns the parent *XMLTag*. If NULL is returned, this *XMLTag* is the root *XMLTag*.

SYNTAX

```
RvStatus RvXMLTagGetParentTag(
    IN  RvXMLTagHandle     hTag,
    OUT RvXMLTagHandle*    phParent);
```

PARAMETERS

hTag

The *XMLTag* whose parent should be returned.

phParent

The parent *XMLTag*.

RETURN VALUES

Returns RvStatus.

RvXMLTagSetName()

DESCRIPTION

Sets the name of the *XMLTag*. The name will be encoded to:

```
<nameStr>tagData<\nameStr>
```

SYNTAX

```
RvStatus RvXMLTagSetName(
    IN RvXMLTagHandle    hTag,
    IN RvChar*           nameStr);
```

PARAMETERS

hTag

The handle to *XMLTag* for which to set the name.

nameStr

The string to set as the name of the tag.

RETURN VALUES

Returns RvStatus.

RvXMLTagGetName()

DESCRIPTION

Returns the name of the *XMLTag*.

SYNTAX

```
RvStatus RvXMLTagGetName(
    IN    RvXMLTagHandle    hTag,
    INOUT RvInt32*          len,
    INOUT RvChar*           nameStr);
```

PARAMETERS

hTag

The handle to *XMLTag* for which to set the name.

len

The length of the string that the application supplies, in which the name of the tag will be returned. If the buffer is too small, the function returns RV_ERROR_INSUFFICIENT_BUFFER and len contains the required buffer length.

nameStr

The string in which the name of the attribute will be returned.

RETURN VALUES

Returns RvStatus.

RvXMLTagSetData()

DESCRIPTION

Sets the name of the *XMLTag*. The name will be encoded to:

```
<tag>dataStr<\tag>
```

SYNTAX

```
RvStatus RvXMLTagSetData(
    IN RvXMLTagHandle    hTag,
    IN RvChar*           dataStr);
```

PARAMETERS

hTag

The handle to *XMLTag* in which to set the data.

dataStr

The string to set as the data of the tag.

RETURN VALUES

Returns RvStatus.

RvXMLTagGetData()

DESCRIPTION

Returns the data of the *XMLTag*.

SYNTAX

```
RvStatus RvXMLTagGetData(
    IN    RvXMLTagHandle    hTag,
    INOUT RvInt32*          len,
    INOUT RvChar*           dataStr);
```

PARAMETERS

hTag

The handle to *XMLTag* in which to set the data.

len

The length of the string that the application supplies, in which the name of the tag will be returned. if the buffer is too small, the function returns RV_ERROR_INSUFFICIENT_BUFFER and len contains the required buffer length.

dataStr

The string in which the data of the tag will be returned.

RETURN VALUES

Returns RvStatus.

RvXMLTagGetAttribute()

DESCRIPTION

Returns an attribute of an *XMLTag*. The attribute is retrieved according to the location argument (first/last/prev/next). The *hCurrAttr* parameter is the current *XMLAttr* from which to get the *XMLAttr* if the location is next/prev.

SYNTAX

```
RvStatus RvXMLTagGetAttribute(
    IN  RvXMLTagHandle      hTag,
    IN  RvXMLAttrHandle     hCurrAttr,
    IN  RvXMLListLocation   eLocation,
    OUT RvXMLAttrHandle*    phAttr);
```

PARAMETERS

hTag

The handle to the *XMLTag* from which to get the attribute.

hCurrAttr

The current *XMLAttr* to which the new *XMLTag* will be added (when the location is next/prev).

eLocation

The location of the attribute in the attributes list of the *XMLTag*.

phAttr

The handle to the *XMLAttr* that was found.

RETURN VALUES

Returns RvStatus.

RvXMLTagGetAttributeByName()

DESCRIPTION

Returns an attribute for an *XMLTag* by the name of the attribute name. If NULL is returned, no attribute with the name given exists for this tag.

SYNTAX

```
RvStatus RvXMLTagGetAttributeByName(
    IN  RvXMLTagHandle      hTag,
    IN  RvChar*             strName,
    OUT RvXMLAttrHandle*    hAttr);
```

PARAMETERS

hTag

The handle to the *XMLTag* from which to get the attribute.

strName

The name of the attribute which should be returned.

phAttr

The handle to the *XMLAttr* that was found.

RETURN VALUES

Returns RvStatus.

RvXMLTagGetDoc()

DESCRIPTION

Returns the *XMLDoc*.

SYNTAX

```
RvStatus RvXMLTagGetDoc(
    IN RvXMLTagHandle      hTag,
    OUT RvXMLDocHandle*    phDoc);
```

PARAMETERS

hTag

The *XMLTag* whose *XMLDoc* should be returned.

phDoc

The *XMLDoc* of the *XMLTag*.

RETURN VALUES

Returns RvStatus.

RvXMLTagCompare()

DESCRIPTION

Compares two *XMLTag*s.

SYNTAX

```
RvBool RvXMLTagCompare(
    IN RvXMLTagHandle    hLeftTag,
    IN RvXMLTagHandle    hRightTag,
    IN RvInt32           flags);
```

PARAMETERS

hLeftTag

The handle to the left-value *XMLTag*.

hRightTag

The handle to the right value *XMLTag*.

flags

The comparison flags, taken from the RvXMLComparisonFlags enumeration which describes the comparison mode. Some combinations of RvXMLComparisonFlags can be bitwise ORed.

REMARKS

Both tag handles need to be initialized before calling this function.

RETURN VALUES

Returns RV_TRUE if both tags are equivalent. Otherwise returns RV_FALSE.

RvXMLTagFindTuple()

DESCRIPTION

Searches a subtree for a tuple (tag) with a given name and attribute/value pair. If the tuple is found, the function returns RV_OK and the *phTag* will point to the tuple handle of the first tag found containing the attribute name/value pair.

SYNTAX

```
RvStatus RvXMLTagFindTuple(
    IN  RvXMLTagHandle     hRootTag,
    IN  RvChar*            tupleName,
    IN  RvChar*            attrName,
    IN  RvChar*            attrValue,
    IN  RvInt32            flags,
    OUT RvXMLTagHandle*    phTag);
```

PARAMETERS

hRootTag

The valid handle to the searched subtree root.

tupleName

A NULL-terminated buffer containing the tuple name.

attrName

A NULL-terminated buffer containing the name of the attribute which is being searched.

attrValue

A NULL-terminated buffer containing the value of the attribute which is being searched.

flags

The comparison flags, taken from the RvXMLComparisonFlags enumeration, which describe the comparison mode. Some combinations of RvXMLComparisonFlags can be bitwise ORed.

phTag

A pointer to an un-initialized tag handle. If the tuple is found and the function returns RV_OK, the phTag will point to the tuple handle. In any other case the phTag will be NULL.

RETURN VALUES

Returns RvStatus.

Upon error the *phTag* will be NULL.

RvXMLTagCopy()

DESCRIPTION

Copies the name/value/attributes of an existing tag to another tag in an *XMLDoc*.

SYNTAX

```
RvStatus RvXMLTagCopy(
    IN RvXMLDocHandle    hSrcDoc,
    IN RvXMLTagHandle    hSourceTag,
    IN RvXMLDocHandle    hTargetDoc,
    IN RvXMLTagHandle    hTargetTag);
```

PARAMETERS

hSrcDoc

The *XMLDoc* that include the source tag.

hSourceTag

The tag whose name/value/attributes are being copied.

hTargetDoc

The *XMLDoc* that includes the target tag.

hTargetTag

The tag to which the name/value/attributes are copied.

RETURN VALUES

Returns RvStatus.

RvXMLTagGetChildTagByNameEx()

DESCRIPTION

Returns a child *XMLTag* of a given parent *XMLTag* by the name and namespace of the child *XMLTag*. If NULL is returned, no child *XMLTag* exists with the given name and namespace.

SYNTAX

```
RvStatus RvXMLTagGetChildTagByNameEx(
    IN  RvXMLTagHandle     hParent,
    IN  RvChar*            strNSPrefix,
    IN  RvChar*            strNS,
    IN  RvChar*            strName,
    OUT RvXMLTagHandle*    phTag);
```

PARAMETERS

hParent

The parent *XMLTag* whose child *XMLTag* should be returned from its list.

strNSPrefix

The namespace prefix of the *XMLTag*.

strNS

The namespace value of the *XMLTag*.

strName

The name of the *XMLTag* which should be returned.

phTag

The returned *XMLTag*.

RETURN VALUES

Returns RvStatus.

RvXMLTagSetNameSpace()

DESCRIPTION

Sets the namespace of the *XMLTag*.

SYNTAX

```
RvStatus RvXMLTagSetNameSpace(
    IN RvXMLTagHandle    hTag,
    IN RvChar*           strNSPrefix,
    IN RvChar*           strNS);
```

PARAMETERS

hTag

The handle to the *XMLTag* for which to set the namespace.

strNSPrefix

The namespace prefix of the *XMLTag*.

strNS

The namespace value of the *XMLTag*.

RETURN VALUES

Returns RvStatus.

RvXMLTagGetNameSpace()

DESCRIPTION

Returns the name of the *XMLTag*.

SYNTAX

```
RvStatus RvXMLTagGetNameSpace(
    IN    RvXMLTagHandle    hTag,
    INOUT RvInt32*          lenNSPrefix,
    OUT   RvChar*           strNSPrefix,
    INOUT RvInt32*          lenNS,
    OUT   RvChar*           strNS);
```

PARAMETERS

hTag

The handle to the *XMLTag* for which to set the name.

lenNSPrefix

A pointer to an integer owned by the caller containing the maximum capacity of *strNSPrefix*. Upon successful completion, the integer will contain the length of the namespace prefix.

strNSPrefix

The pre-allocated buffer owned by the caller. Upon successful completion, the buffer will contain the namespace prefix.

lenNS

A pointer to an integer owned by the caller containing the maximum capacity of *strNS*. Upon successful completion, the integer will contain the length of the namespace value.

strNS

The pre-allocated buffer owned by the caller. Upon successful completion, the buffer will contain the namespace value.

RETURN VALUES

Returns RvStatus.

RvXMLTagGetAttributeByNameEx()

DESCRIPTION

Returns an attribute for an *XMLTag* by the attributes name. If NULL is returned, no attribute with the name given exists for this *XMLTag*.

SYNTAX

```
RvStatus RvXMLTagGetAttributeByNameEx(
    IN  RvXMLTagHandle      hTag,
    IN  RvChar*             strName,
    IN  RvChar*             strNSPrefix,
    IN  RvChar*             strNS,
    OUT RvXMLAttrHandle*    phAttr);
```

PARAMETERS

hTag

The handle to the *XMLTag* from which to get the attribute.

strName

The name of the attribute that should be returned.

strNSPrefix

The namespace prefix of the *XMLTag*.

strNS

The namespace value of the *XMLTag*.

phAttr

The handle to the *XMLAttr* that was found.

RETURN VALUES

Returns RvStatus.

RvXMLTagFindTupleEx()

DESCRIPTION

Searches a subtree for a tuple (tag) with a given name and attribute/value pair. If the tuple is found, the function returns RV_OK and *phTag* will point to the tuple handle of the first tag found containing the attribute name/value pair.

SYNTAX

```
RvStatus RvXMLTagFindTupleEx(
    IN  RvXMLTagHandle     hRootTag,
    IN  RvChar*            strTupleNSPrefix,
    IN  RvChar*            strTupleNS,
    IN  RvChar*            strTupleName,
    IN  RvChar*            strAttrNSPrefix,
    IN  RvChar*            strAttrNS,
    IN  RvChar*            strAttrName,
    IN  RvChar*            strAttrValue,
    IN  RvInt32            flags,
    OUT RvXMLTagHandle*    phTag);
```

PARAMETERS

hRootTag

The valid handle to the searched subtree root.

strTupleNSPrefix

The NULL-terminated buffer containing the namespace prefix of the tuple (tag).

strTupleNS

The NULL-terminated buffer containing the namespace value of the tuple (tag).

strTupleName

The NULL-terminated buffer containing the tuple name.

strAttrNSPrefix

The NULL-terminated buffer containing the namespace prefix of the attribute.

strAttrNS

The NULL-terminated buffer containing the namespace value of the attribute.

strAttrName

The NULL-terminated buffer containing the name of the attribute which is being searched.

strAttrValue

The NULL-terminated buffer containing the value of the attribute which is being searched.

flags

Comparison flags, taken from the RvXMLComparisonFlags enumeration which describe the comparison mode. Some combinations of RvXMLComparisonFlags can be bitwise ORed.

phTag

A pointer to an un-initialized tag handle. If the tuple is found and the function returns RV_OK, *phTag* will point to the tuple handle. In any other case, *phTag* will be NULL.

RETURN VALUES

Returns RvStatus.

RvXMLTagSetType()

DESCRIPTION

Sets the type of the *XMLTag*.

SYNTAX

```
RvStatus RvXMLTagSetType(
    IN RvXMLTagHandle    hTag,
    IN RvXMLNodeType     tagType);
```

PARAMETERS

hTag

The handle to the *XMLTag* for which to set the name.

tagType

The node type can be any one of the values from RvXmlNodeType enumeration.

RETURN VALUES

Returns RvStatus.

RvXMLTagGetType()

DESCRIPTION

Retrieves the type of the *XMLTag*.

SYNTAX

```
RvStatus RvXMLTagGetType(
    IN  RvXMLTagHandle    hTag,
    OUT RvXMLNodeType*    pTagType);
```

PARAMETERS

hTag

The handle to the *XMLTag* for which to set the name.

pTagType

A pointer to an RvXmlNodeType item. Upon successful completion, this parameter will contain a value describing the node type.

RETURN VALUES

Returns RvStatus.

RvXMLTagNSLocalToGlobal()

DESCRIPTION

Redefines the local default namespace of the tag (attribute: `"xmlns=xxx"`) to a global namespace. For example, for a name such as "nsN", the attribute of the root tag will be: `"xmlns:nsN=xxx"`. Set this new namespace for all descendant nodes.

SYNTAX

```
RvStatus RvXMLTagNSLocalToGlobal(
    IN RvXMLTagHandle    hTag);
```

PARAMETERS

hTag

The handle to the *XMLTag* to redefine local default namespace.

RETURN VALUES

Returns RvStatus.

RvXMLGetNamespaceAttr()

DESCRIPTION

Returns an attribute that defines the namespace with a specified prefix and/or namespace value, such as `xmlns:<prefix>="<namespace>"`. If NULL is returned, no attribute with the given prefix/namespace exists for this *XMLTag*.

SYNTAX

```
RvStatus RvXMLGetNamespaceAttr(
    IN  RvXMLTagHandle      hTag,
    IN  RvChar*             strNSPrefix,
    IN  RvChar*             strNSValue,
    OUT RvXMLAttrHandle*    phAttr);
```

PARAMETERS

hTag

The handle to the *XMLTag* from which to get the attribute.

strNSPrefix

The namespace prefix.

strNSValue

The namespace value.

phAttr

The handle to the *XMLAttr* that was found (NULL, if not found).

RETURN VALUES

Returns RvStatus.

RvXMLGetNamespaceAttrEx()

DESCRIPTION

Returns an attribute that defines the namespace with a specified prefix and/or namespace value, such as `xmlns:<prefix>="<namespace>"`. If NULL is returned, no attribute with the given prefix/namespace exists for this *XMLTag* and for any ascendant tags.

SYNTAX

```
RvStatus RvXMLGetNamespaceAttrEx(
    IN  RvXMLTagHandle      hTag,
    IN  RvChar*             strNSPrefix,
    IN  RvChar*             strNSValue,
    OUT RvXMLAttrHandle*    phAttr);
```

PARAMETERS

hTag

The handle to the *XMLTag* from which to get the attribute.

strNSPrefix

The namespace prefix.

strNSValue

The namespace value.

phAttr

The handle to the *XMLAttr* that was found (NULL, if not found).

Note: The returned attribute may not belong to the given tag, but to its parent (ascendant) tag, including the root tag.

RETURN VALUES

Returns RvStatus.

RvXMLGetDefaultNamespaceAttr()

DESCRIPTION

Returns an attribute that defines the namespace which is the default namespace for the given *XMLTag*. If NULL is returned, the default namespace is not defined for this tag.

SYNTAX

```
RvStatus RvXMLGetDefaultNamespaceAttr(
    IN  RvXMLTagHandle      hTag,
    OUT RvXMLAttrHandle*    phAttr);
```

PARAMETERS

hTag

The handle to the *XMLTag* from which to get the attribute.

phAttr

The handle to the *XMLAttr* that was found (NULL, if not found).

Note: The returned attribute may not belong to the given tag, but to its parent (ascendant) tag, including the root tag.

RETURN VALUES

Returns RvStatus.

RvXMLCreateNamespaceAttr()

DESCRIPTION

Creates an attribute that defines the namespace with a specified prefix and namespace value, such as `xmlns:<prefix>="<namespace>"`.

SYNTAX

```
RvStatus RvXMLCreateNamespaceAttr(
    IN  RvXMLTagHandle      hTag,
    IN  RvChar*             strNSPrefix,
    IN  RvChar*             strNSValue,
    OUT RvXMLAttrHandle*    phAttr);
```

PARAMETERS

hTag

The handle to the *XMLTag* to create the child attribute.

strNSPrefix

The namespace prefix.

strNSValue

The namespace value.

phAttr

The handle to the *XMLAttr* that was created.

RETURN VALUES

Returns RvStatus.

RvXMLTagNameCompareWns()

DESCRIPTION

Compares the name, namespace prefix, and namespace of the given tag. RV_ERROR_NOT_FOUND means that the name and/or prefix and/or namespace are not equal to the specified values.

SYNTAX

```
RvStatus RvXMLTagNameCompareWns(
    IN RvXMLTagHandle    hTag,
    IN RvChar*           strName,
    IN RvChar*           strNSPrefix,
    IN RvChar*           strNSValue);
```

PARAMETERS

hTag

The handle to the *XMLTag*.

strName

The string to compare with the *XMLTag* name.

strNSPrefix

The string to compare with the *XMLTag* namespace prefix.

strNSValue

The string to compare with the *XMLTag* namespace value.

Notes:

- Case-sensitive comparing is performed.
- Any input string can be NULL. Comparing is not performed in this case.

RETURN VALUES

Returns RvStatus.

RvXMLTagNextDescendant()

DESCRIPTION

Enumerates all descendant tags of the given *XMLTag*.

SYNTAX

```
RvStatus RvXMLTagNextDescendant(
    IN    RvXMLTagHandle     hTag,
    INOUT RvXMLTagHandle*    phTagDesc);
```

PARAMETERS

hTag

The handle to the *XMLTag*.

phTagDesc

A pointer to the variable that receives the *XMLTag* handle of the next descendant tag or NULL, if no more.

Notes:

- To begin enumerating all sub-tags, the caller has to set the value pointed by *phTagDesc* to NULL.
- The caller must not change the value pointed by *phTagDesc* between sequential calls to this function.
- The function returns NULL in *phTagDesc* if no more descendant tags exist.

RETURN VALUES

Returns RvStatus.

RvXMLTagEncode()

DESCRIPTION

Encodes an *XMLTag* to a string. Each *XMLTag* will be represented as `<TagName>data</TagName>`. Recursively, all the children of the *XMLTag* will be inserted. Each *XMLAttr* will be added to the tag in the format of:

`<TagName AttrName="attribute value"></TagName>.`

SYNTAX

```
RvStatus RvXMLTagEncode(
    IN    RvXMLTagHandle    hTag,
    INOUT RvInt32*          len,
    INOUT RvChar*           pTagStr);
```

PARAMETERS

hTag

The handle to the *XMLTag* to encode.

len

The length of the string supplied by the application to which to encode the *XMLTag*. If the buffer is too small, the function returns RV_ERROR_INSUFFICIENT_BUFFER and *len* contains the required buffer length.

pTagStr

The string in which the *XMLTag* will be encoded.

RETURN VALUES

Returns RvStatus.

XML Document Functions

An XML Document object (*XMLDoc*) represents an XML document as a tree of XML tags (*XMLTag*). The XML Document API functions allow you to create and destruct *XMLDoc* objects, and encode an *XMLDoc* to an XML document that is represented as a string. This section contains the XML Document API functions found in the *RvXMLDoc.h* header file.

The XML Document API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvXMLDocCreate()
- RvXMLDocDestruct()
- RvXMLDocEncode()
- RvXMLDocCopy()

RvXMLDocCreate()

DESCRIPTION

Creates a new *XMLDoc* by the allocation of an object in the *XMLDoc* array. The header of the *XMLDoc* will be created and initialized to the default value:

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

SYNTAX

```
RvStatus RvXMLDocCreate(
    IN  RvXMLMgrHandle      hMgr,
    OUT RvXMLDocHandle*     phDoc);
```

PARAMETERS

hMgr

The handle to the *XMLMgr*.

phDoc

The handle to the *XMLDoc* that was created.

RETURN VALUES

Returns RvStatus.

RvXMLDocDestruct()

DESCRIPTION

Destructs an *XMLDoc*. All the *XMLTag* and *XMLAttr* objects of the *XMLDoc* will be destructed. The memory page of this *XMLDoc* will be released.

SYNTAX

```
RvStatus RvXMLDocDestruct(
    IN RvXMLDocHandle    hDoc);
```

PARAMETERS

hDoc

The handle to the *XMLDoc* to be destructed.

RETURN VALUES

Returns RvStatus.

RvXMLDocEncode()

DESCRIPTION

Encodes an *XMLDoc* to a string. Each *XMLTag* will be represented as:

```
<TagName>data</TagName>
```

Recursively, all the children of the *XMLTag* will be inserted. Each *XMLAttr* will be added to the tag in the following format:

```
<TagName AttrName="attribute value"></TagName>
```

The header of the *XMLDoc* will be added at the beginning of the document.

SYNTAX

```
RvStatus RvXMLDocEncode(
    IN    RvXMLDocHandle    hDoc,
    INOUT RvInt32*          len,
    INOUT RvChar*           pDocStr);
```

PARAMETERS

hDoc

The handle to the *XMLDoc* to encode.

len

The length of the string that the application supplies, in which to encode the *XMLDoc*. If the buffer is too small, the function returns RV_ERROR_INSUFFICIENT_BUFFER and *len* contains the required buffer length.

pDocStr

The string where the *XMLDoc* will be encoded.

RETURN VALUES

Returns RvStatus.

RvXMLDocCopy()

DESCRIPTION

Copies an existing *XMLDoc* to a new document.

SYNTAX

```
RvStatus RvXMLDocCopy(
    IN    RvXMLDocHandle    hSrcDoc,
    INOUT RvXMLDocHandle    hTargetDoc);
```

PARAMETERS

hSrcDoc

The handle to the *XMLDoc* that needs to be copied.

hTargetDoc

The handle to the copied *XMLDoc*. The *hTargetDoc* handle should be created before calling this function (using RvXMLDocCreate()).

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

The Get and Set functions are:

- RvXMLDocGetRootTag()
- RvXMLDocGetTagByName()
- RvXMLDocSetHeaderName()
- RvXMLDocGetHeaderName()
- RvXMLDocSetHeaderData()
- RvXMLDocGetHeaderData()
- RvXMLDocGetXMLMgr()
- RvXMLDocGetHeaderTag()
- RvXMLDocCopySubTree()
- RvXMLDocCompareSubTree()
- RvXMLDocFindTuple()
- RvXMLDocRemoveSubTree()
- RvXMLDocGetTagByNameEx()
- RvXMLDocFindTupleEx()
- RvXMLDocCopySubTreeEx()
- RvXMLDocParse()

RvXMLDocGetRootTag()

DESCRIPTION

Returns a handle to the root *XMLTag* in the *XMLDoc*. This is the first tag in the hierarchy, with no parent tags. This function should be called after constructing a new *XMLDoc* for setting the name and data of the root tag.

SYNTAX

```
RvStatus RvXMLDocGetRootTag(
    IN  RvXMLDocHandle     hDoc,
    OUT RvXMLTagHandle*    hTag);
```

PARAMETERS

hDoc

The handle to the *XMLDoc* from which to get the root tag.

phTag

The root *XMLTag* of the *XMLDoc*.

RETURN VALUES

Returns RvStatus.

RvXMLDocGetTagByName()

DESCRIPTION

Returns an *XMLTag* for a given *XMLDoc* by the name of the tag. If NULL is returned, no child tag with the given name exists for this document.

SYNTAX

```
RvStatus RvXMLDocGetTagByName(
    IN  RvXMLDocHandle     hDoc,
    IN  RvChar*            strName,
    OUT RvXMLTagHandle*    phTag);
```

PARAMETERS

hDoc

The handle to the *XMLDoc* from which to get the root tag.

strName

The name of the tag that should be returned.

phTag

The root *XMLTag* of the *XMLDoc*.

RETURN VALUES

Returns RvStatus.

RvXMLDocSetHeaderName()

DESCRIPTION

The header of the *XMLDoc* is the first tag in the *XMLDoc* that is not recursive (meaning that it does not have any child *XMLTag* objects). For example:

```
<?xml version="1.0"?>
```

This functions sets the header name. (In this example, the header name is "`xml`").

SYNTAX

```
RvStatus RvXMLDocSetHeaderName(
    IN RvXMLDocHandle    hDoc,
    IN RvChar*           nameStr);
```

PARAMETERS

hDoc

The handle to *XMLDoc* tin which to set the name of the header.

nameStr

The string to set as the name of the header.

RETURN VALUES

Returns RvStatus.

RvXMLDocGetHeaderName()

DESCRIPTION

Returns the name of the header.

SYNTAX

```
RvStatus RvXMLDocGetHeaderName(
    IN    RvXMLDocHandle    hDoc,
    INOUT RvInt32*          len,
    INOUT RvChar*           nameStr);
```

PARAMETERS

hDoc

The handle to *XMLDoc* from which to get the name of the header.

len

The length of the string that the application supplies, in which the name of the header will be returned. If the buffer is too small, the function returns RV_ERROR_INSUFFICIENT_BUFFER and len contains the required buffer length.

nameStr

The string in which the name of the header will be returned.

RETURN VALUES

Returns RvStatus.

RvXMLDocSetHeaderData()

DESCRIPTION

The header of the *XMLDoc* is the first tag in the *XMLDoc* which is not recursive (meaning that it does not have any child *XMLTag* objects). For example:

```
<?xml version="1.0"?>
```

This function sets the header data. (In this example, the version is "1.0").

SYNTAX

```
RvStatus RvXMLDocSetHeaderData(
    IN RvXMLDocHandle    hDoc,
    IN RvChar*           dataStr);
```

PARAMETERS

hDoc

The handle to the *XMLDoc* in which to set the data of the header.

dataStr

The string to set as the data of the header.

RETURN VALUES

Returns RvStatus.

RvXMLDocGetHeaderData()

DESCRIPTION

Returns the data of the header.

SYNTAX

```
RvStatus RvXMLDocGetHeaderData(
    IN    RvXMLDocHandle    hDoc,
    INOUT RvInt32*          len,
    INOUT RvChar*           dataStr);
```

PARAMETERS

hDoc

The handle to the *XMLDoc* from which to get the data of the header.

len

The length of the string that the application supplies, in which the data of the header will be returned. If the buffer is too small, the function returns RV_ERROR_INSUFFICIENT_BUFFER and len contains the required buffer length.

dataStr

The string in which the data of the header will be returned.

RETURN VALUES

Returns RvStatus.

RvXMLDocGetXMLMgr()

DESCRIPTION

Returns the *XMLMgr* from the *XMLDoc*.

SYNTAX

```
RvStatus RvXMLDocGetXMLMgr(
    IN  RvXMLDocHandle     hDoc,
    OUT RvXMLMgrHandle*    phMgr);
```

PARAMETERS

hDoc

The handle to the *XMLDoc*.

phMgr

The handle to the *XMLMgr*.

RETURN VALUES

Returns RvStatus.

RvXMLDocGetHeaderTag()

DESCRIPTION

Returns a handle to the header *XMLTag* in the *XMLDoc*. This is the processing-instruction tag, such as `<?xml ... ?>`.

SYNTAX

```
RvStatus RvXMLDocGetHeaderTag(
    IN  RvXMLDocHandle     hDoc,
    OUT RvXMLTagHandle*    phTag);
```

PARAMETERS

hDoc

The handle to the *XMLDoc* from which to get the root tag.

phTag

The header *XMLTag* of the *XMLDoc*.

RETURN VALUES

Returns RvStatus.

RvXMLDocCopySubTree()

DESCRIPTION

Copy a subtree from a source *XMLDoc* to a target *XMLDoc* The tag itself (*hTargetParentTag*) should be constructed outside the function. The function copies all the attributes/tags of *hSourceParentTag* in recursion.

SYNTAX

```
RvStatus RvXMLDocCopySubTree(
    IN RvXMLDocHandle    hSrcDoc,
    IN RvXMLDocHandle    hTargetDoc,
    IN RvXMLTagHandle    hSourceParentTag,
    IN RvXMLTagHandle    hTargetParentTag);
```

PARAMETERS

hSourceDoc

The source *XMLDoc*.

hTargetDoc

The target *XMLDoc*.

hSourceParentTag

The *XMLTag* which is the parent of the *XMLTag* that needs to be copied.

hTargetParentTag

The *XMLTag* which is the parent of the new *XMLTag* in the target *XMLDoc*. This *XMLTag* is added to the *XMLTag* list of the parent of the target.

RETURN VALUES

Returns RvStatus.

RvXMLDocCompareSubTree()

DESCRIPTION

Compares a subtree between “left” and “right” *XMLDoc* objects. The tag handles are the roots of the subtrees the caller wishes to compare. This function recurses through the subtrees until it either finds differences or it completely traverses the subtrees, whichever happens first.

SYNTAX

```
RvBool RvXMLDocCompareSubTree(
    IN RvXMLDocHandle    hLeftDoc,
    IN RvXMLDocHandle    hRightDoc,
    IN RvXMLTagHandle    hRootLeftSubtree,
    IN RvXMLTagHandle    hRootRightSubtree,
    IN RvInt32           flags);
```

PARAMETERS

hLeftDoc

The handle to the left *XMLDoc*.

hRightDoc

The handle to the right *XMLDoc*.

hRootLeftSubtree

The handle to the root *XMLTag* of the left subtree which is to be compared.

hRootRightSubtree

The handle to the root *XMLTag* of the right subtree which is to be compared.

flags

The comparison flags taken from the RvXMLComparisonFlags enumeration, which describe the comparison mode. Some combinations of RvXMLComparisonFlags can be bitwise ORed.

RETURN VALUES

Returns RV_TRUE if both tags are equivalent. Otherwise, returns RV_FALSE if both attributes are different, or bad Parameters are passed to the function.

RvXMLDocFindTuple()

DESCRIPTION

Searches an *XMLDoc* for a tuple (tag) with a given name and attribute/value pair. If the tuple is found, the function returns RV_OK and phTag will point to the handle of the first tuple found containing the attribute name/value pair.

SYNTAX

```
RvStatus RvXMLDocFindTuple(
    IN  RvXMLDocHandle     hDoc,
    IN  RvChar*            tupleName,
    IN  RvChar*            attrName,
    IN  RvChar*            attrValue,
    IN  RvInt32            flags,
    OUT RvXMLTagHandle*    phTag);
```

PARAMETERS

hDoc

The valid handle to the source *XMLDoc* that is being searched.

tupleName

The NULL-terminated buffer containing the tuple name.

attrName

The NULL-terminated buffer containing the name of the attribute that is being searched.

attrValue

The NULL-terminated buffer containing the value of the attribute that is being searched.

flags

The comparison flags taken from RvXMLComparisonFlags enumeration, which describe the comparison mode. Some combinations of RvXMLComparisonFlags can be bitwise ORed.

phTag

A pointer to an un-initialized RvXMLTagHandle. If the tuple is found and the function returns RV_OK, *phTag* will point to the tuple handle. In any other case, *phTag* will point to NULL.

RETURN VALUES

Returns RvStatus.

RvXMLDocRemoveSubTree()

DESCRIPTION

Removes a subtree from an *XMLDoc*.

SYNTAX

```
RvStatus RvXMLDocRemoveSubTree(
    IN RvXMLDocHandle    hDoc,
    IN RvXMLTagHandle    hSubTreeTag);
```

PARAMETERS

hDoc

The valid handle to the source *XMLDoc* from which the subtree is being deleted.

hSubTreeTag

The valid handle to the root tag of the subtree that needs to be removed from the *XMLDoc*.

REMARKS

If *hSubTreeTag* is the root tag, the *XMLDoc* will be destroyed and *hDoc* will become unusable. After a call to this function, *hSubTreeTag* is invalid and it is the responsibility of the caller not to continue using it any further. The removed subtree is destroyed and all associated resources are freed.

RETURN VALUES

Returns RvStatus.

RvXMLDocGetTagByNameEx()

DESCRIPTION

Returns an *XMLTag* for a given *XMLDoc* by the name of the *XMLTag*. If NULL is returned, no child tag with the given name exists for this *XMLDoc*.

SYNTAX

```
RvStatus RvXMLDocGetTagByNameEx(
    IN  RvXMLDocHandle     hDoc,
    IN  RvChar*            strNSPrefix,
    IN  RvChar*            strNS,
    IN  RvChar*            strName,
    OUT RvXMLTagHandle*    phTag);
```

PARAMETERS

hDoc

The handle to the *XMLDoc* from which to get the root *XMLTag*.

strNSPrefix

The NULL-terminated string containing the namespace prefix.

strNS

The NULL-terminated string containing the namespace value.

strName

The NULL-terminated string containing the name of the tag which should be returned.

phTag

The the root *XMLTag* of the *XMLDoc*.

RETURN VALUES

Returns RvStatus.

RvXMLDocFindTupleEx()

DESCRIPTION

Searches an *XMLDoc* for a tuple (tag) with a given name and attribute/value pair. If the tuple is found the function returns RV_OK and *phTag* will point to the handle of the first tuple found containing the attribute name/value pair. The search is aware of the namespace of the tuple and attribute.

SYNTAX

```
RvStatus RvXMLDocFindTupleEx(
    IN  RvXMLDocHandle     hDoc,
    IN  RvChar*            strTupleNSPrefix,
    IN  RvChar*            strTupleNS,
    IN  RvChar*            strTupleName,
    IN  RvChar*            strAttrNSPrefix,
    IN  RvChar*            strAttrNS,
    IN  RvChar*            strAttrName,
    IN  RvChar*            strAttrValue,
    IN  RvInt32            flags,
    OUT RvXMLTagHandle*    phTag);
```

PARAMETERS

hDoc

The valid handle to the source *XMLDoc* which is being searched.

strTupleNSPrefix

The NULL-terminated buffer containing the namespace prefix of the tuple (tag).

strTupleNS

The NULL-terminated buffer containing the namespace value of the tuple (tag).

strTupleName

The NULL-terminated buffer containing the tuple name.

strAttrNSPrefix

The NULL-terminated buffer containing the namespace prefix of the attribute.

strAttrNS

The NULL-terminated buffer containing the namespace value of the attribute.

strAttrName

The NULL-terminated buffer containing the name of the attribute which is being searched.

strAttrValue

The NULL-terminated buffer containing the value of the attribute which is being searched.

flags

The comparison flags, taken from RvXMLComparisonFlags enumeration which describe the comparison mode. Some combinations of RvXMLComparisonFlags can be bitwise ORed.

phTag

A pointer to an un-initialized RvXMLTagHandle. If the tuple is found and the function returns RV_OK, *phTag* will point to the tuple handle. In any other case, *phTag* will point to NULL.

Return Values

Returns RvStatus.

RvXMLDocCopySubTreeEx()

DESCRIPTION

Copies a subtree from a source *XMLDoc* to a target *XMLDoc*. The tag itself (*hTargetParentTag*) should be constructed outside the function. The function copies all the attributes/tags of *hSourceParentTag* in recursion.

SYNTAX

```
RvStatus RvXMLDocCopySubTreeEx(
    IN RvXMLDocHandle       hSourceDoc,
    IN RvXMLDocHandle       hTargetDoc,
    IN RvXMLTagHandle       hSourceParentTag,
    IN RvXMLTagHandle       hTargetParentTag,
    IN RvXMLTagHandle       hTargetLocationTag,
    IN RvXMLListLocation    eLocation);
```

PARAMETERS

hSourceDoc

The source *XMLDoc*.

hTargetDoc

The target *XMLDoc*.

hSourceParentTag

The node of the source *XMLDoc*. All descendant nodes of this node are to be copied to the target *XMLDoc*.

hTargetParentTag

The node that is the parent for all the tags that need to be copied.

hTargetLocationTag

The child node of the *hTargetParentTag* to which the new nodes will be added (next/prev).

eLocation

The location of the new nodes (first/last/next/prev).

REMARKS

If the prefix of the tag points to a namespace which equals the namespace in the target (`r->pidf in src` and `r->pidf` in target), simply copy the tag. If the prefix of the tag points to the namespace which differs from the namespace in the target:

- Option A: The target has no definition of the namespace: (`r->pidf in src and r->rpid` in target and no prefix in the dest of pidf).

  In this case, `add r1->pidf` to the target and replace all appearances of "r" in the copied element to "r1".

- Option B: The target has another definition of the namespace: (`r->pidf in src and q->pidf in target`).

  In this case, replace all appearances of "r" in the copied element to "q". If the prefix of the tag points to the namespace which does not appear in target, add the namespace declaration to the target and simply copy the tag. If the tag includes the local namespace declaration, add them to the root of the target doc (according to above rules) If there is no prefix to the tag but the default namespace points to the namespace which differs from the default namespace in the target, add the namespace to the root of the target with some prefix and copy the tag with the new prefix. The source document (*including hSourceParentTag*) is kept unchanged.

RETURN VALUES

Returns RvStatus.

RvXMLDocParse()

DESCRIPTION

Parses a given buffer into an *XMLDoc*.

SYNTAX

```
RvStatus RvXMLDocParse(
    IN RvXMLDocHandle    hDoc,
    IN const RvChar*     pBuffer,
    IN RvUint32          bufLen);
```

PARAMETERS

hDoc

The *XMLDoc* that will hold the parsed XML.

pBuffer

The buffer that contains the XML.

bufLen

The length of the buffer.

REMARKS

The *hDoc* should be created with RvXMLDocCreate() before calling this function. The buffer must contain well-formed XML, otherwise an error will be returned. The parser removes insignificant whitespace from the document.

RETURN VALUES

Returns RvStatus.

XML Attribute Functions

An XML Attribute object (*XMLAttr*) represents an attribute in an XML Tag (*XMLTag*). The *XMLAttr* is identified by its name and value which must be supplied. The XML Attribute API functions allow you to create and destruct *XMLAttr* objects for an *XMLTag*. This section contains the XML Attribute API functions found in the *RvXMLAttr.h* header file.

The XML Attribute API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvXMLAttrCreate()
- RvXMLAttrCreateEx()
- RvXMLAttrCompare()
- RvXMLAttrNameCompareWns()
- RvXMLAttrDestruct()

RvXMLAttrCreate()

DESCRIPTION

Creates a new *XMLAttr* in an *XMLTag*. The *XMLAttr* will be added to the attributes list of the *XMLTag* according to the *eLocation* parameter (first/last/next/prev). The *hCurrAttr* parameter will be used as a reference when the location is next/prev. The name and value can be supplied and will be used for encoding in the following format:

```
<XMLTag strName="strValue"><\XMLTag>
```

If the name and value Parameters are not supplied, they must be set by the application with the RvXMLAttrSetName() and RvXMLAttrSetValue() functions.

SYNTAX

```
RvStatus RvXMLAttrCreate(
    IN  RvXMLTagHandle       hTag,
    IN  RvXMLAttrHandle      hCurrAttr,
    IN  RvXMLListLocation    eLocation,
    IN  RvChar*              strName,
    IN  RvChar*              strValue,
    OUT RvXMLAttrHandle*     phAttr);
```

PARAMETERS

hTag

The handle to the *XMLTag* in which to create the new *XMLAttr*.

hCurrAttr

The current *XMLAttr* to which the new *XMLAttr* will be added (when the location is next/prev).

eLocation

The location of the new *XMLAttr* (first/last/next/prev).

strName

The name of the new *XMLAttr*.

strValue

The value of the new *XMLAttr*.

phAttr

The handle to the new *XMLAttr* that was created.

RETURN VALUES

Returns RvStatus.

RvXMLAttrCreateEx()

DESCRIPTION

Creates a new *XMLAttr* in an *XMLTag* and associates an XML namespace with the *XMLAttr*. If no namespace is specified (NULL), this function is equivalent to RvXMLDocCreate(). The *XMLAttr* will be added to the *XMLTag* attributes list according to the location argument (first/last/next/prev). The value for *hCurrAttr* will be used as reference when the location is next/prev. The name and value can be supplied and will be used for encoding in the following format:

`<XMLTag strName="strValue"><\XMLTag>.`

If the name and value parameters are not supplied, they must be set by the application with the RvXMLAttrSetName() and RvXMLAttrSetValue() functions.

SYNTAX

```
RvStatus RvXMLAttrCreateEx(
    IN  RvXMLTagHandle       hTag,
    IN  RvXMLAttrHandle      hCurrAttr,
    IN  RvXMLListLocation    eLocation,
    IN  RvChar*              strNSPrefix,
    IN  RvChar*              strNS,
    IN  RvChar*              strName,
    IN  RvChar*              strValue,
    OUT RvXMLAttrHandle*     phAttr);
```

PARAMETERS

hTag

The handle to the *XMLTag* in which to create the new *XMLAttr*.

hCurrAttr

The current *XMLAttr* to which the new *XMLAttr* will be added (next/prev).

eLocation

The location of the new *XMLAttr* (first/last/next/prev).

strNSPrefix

The NULL-terminated string containing the namespace prefix.

strNS

The NULL-terminated string containing the namespace value.

strName

The NULL-terminated string containing the attribute name.

strValue

The NULL-terminated string containing the attribute value.

phAttr

The handle to the newly created *XMLAttr*.

RETURN VALUES

Returns RvStatus.

RvXMLAttrCompare()

DESCRIPTION

Compares two *XMLAttr*s in an *XMLTag*.

SYNTAX

```
RvBool RvXMLAttrCompare(
    IN RvXMLAttrHandle    hLeftAttribute,
    IN RvXMLAttrHandle    hRightAttribute,
    IN RvInt32            flags);
```

PARAMETERS

hLeftAttribute

The handle to the left *XMLAttr* value.

hRightAttribute

The handle to the right *XMLAttr* value.

flags

The comparison flags taken from the RvXMLComparisonFlags enumeration, which describe the comparison mode.

RETURN VALUES

Returns RV_TRUE if both tags are equivalent. Otherwise, returns RV_FALSE.

REMARKS

Both attribute handles need to be initialized beforehand.

RvXMLAttrNameCompareWns()

DESCRIPTION

Compares the name, namespace prefix, and namespace of the given *XMLAttr*.

SYNTAX

```
RvStatus RvXMLAttrNameCompareWns(
    IN RvXMLAttrHandle    hAttr,
    IN RvChar*            strName,
    IN RvChar*            strNSPrefix,
    IN RvChar*            strNS);
```

PARAMETERS

hAttr

The handle to the *XMLAttr*.

strName

The string to compare with the *XMLAttr* name.

strNSPrefix

The string to compare with the *XMLAttr* namespace prefix.

strNSValue

The string to compare with the *XMLAttr* namespace value.

RETURN VALUES

Returns RvStatus.

REMARKS

- Case-sensitive comparing is performed.
- Any input string can be NULL. Comparing is not performed in this case.

RvXMLAttrDestruct()

DESCRIPTION

Destructs an *XMLAttr*. The *XMLAttr* will be removed from the attributes list of the *XMLTag*.

SYNTAX

```
RvStatus RvXMLAttrDestruct(
    IN RvXMLAttrHandle    hAttr);
```

PARAMETERS

hAttr

The handle to the *XMLAttr* to destruct.

RETURN VALUES

Returns RvStatus.

Get and Set Functions

The Get and Set functions are:

- RvXMLAttrGetTag()
- RvXMLAttrSetName()
- RvXMLAttrGetName()
- RvXMLAttrSetValue()
- RvXMLAttrGetValue()
- RvXMLAttrSetNameSpace()
- RvXMLAttrGetNameSpace()

RvXMLAttrGetTag()

DESCRIPTION

Returns the *XMLTag* that contains the *XMLAttr*.

SYNTAX

```
RvStatus RvXMLAttrGetTag(
    IN  RvXMLAttrHandle    hAttr,
    OUT RvXMLTagHandle*    phTag);
```

PARAMETERS

hAttr

The *XMLAttr* whose *XMLTag* should be returned.

phTag

The *XMLTag* that was returned.

RETURN VALUES

Returns RvStatus.

RvXMLAttrSetName()

DESCRIPTION

Sets the name of the *XMLAttr*. The name will be encoded to:

```
<tag nameStr="value"><\tag>
```

SYNTAX

```
RvStatus RvXMLAttrSetName(
    IN RvXMLAttrHandle    hAttr,
    IN RvChar*            nameStr);
```

PARAMETERS

hAttr

The handle to the *XMLAttr* whose name is to be set.

nameStr

The string to set as the name of the attribute. It must be a NULL-terminated string.

RETURN VALUES

Returns RvStatus.

RvXMLAttrGetName()

DESCRIPTION

Returns the name of the *XMLAttr*.

SYNTAX

```
RvStatus RvXMLAttrGetName(
    IN    RvXMLAttrHandle    hAttr,
    INOUT RvInt32*           len,
    INOUT RvChar*            nameStr);
```

PARAMETERS

hAttr

The handle to *XMLAttr* from which to get the name.

len

The length of the string that the application supplies, in which the name of the attribute will be returned. If the buffer is too small, the function returns RV_ERROR_INSUFFICIENT_BUFFER and len contains the required buffer length.

nameStr

The string in which the name of the attribute will be returned.

RETURN VALUES

Returns RvStatus.

RvXMLAttrSetValue()

DESCRIPTION

Sets the value of the *XMLAttr* objects. The value will be encoded to:

```
<tag name="valueStr"><\tag>
```

SYNTAX

```
RvStatus RvXMLAttrSetValue(
    IN RvXMLAttrHandle    hAttr,
    IN RvChar*            valueStr);
```

PARAMETERS

hAttr

The handle to the *XMLAttr* in which to set the value.

valueStr

The string to set as the value of the attribute. It must be a NULL terminated string.

RETURN VALUES

Returns RvStatus.

RvXMLAttrGetValue()

DESCRIPTION

Returns the value of the *XMLAttr*.

SYNTAX

```
RvStatus RvXMLAttrGetValue(
    IN    RvXMLAttrHandle    hAttr,
    INOUT RvInt32*           len,
    INOUT RvChar*            valueStr);
```

PARAMETERS

hAttr

The handle to the *XMLAttr* from which to get the value.

len

The length of the string that the application supplies, in which the name of the attribute will be returned. If the buffer is too small, the function returns RV_ERROR_INSUFFICIENT_BUFFER and len contains the required buffer length.

valueStr

The string in which the value of the attribute will be returned.

RETURN VALUES

Returns RvStatus.

RvXMLAttrSetNameSpace()

DESCRIPTION

Sets the namespace of the *XMLAttr*.

SYNTAX

```
RvStatus RvXMLAttrSetNameSpace(
    IN RvXMLAttrHandle    hAttr,
    IN RvChar*            strNSPrefix,
    IN RvChar*            strNS);
```

PARAMETERS

hAttr

The handle to the *XMLAttr* for which to set the namespace.

strNSPrefix

The NULL terminated string containing the namespace prefix.

strNS

The NULL terminated string containing the namespace value.

RETURN VALUES

Returns RvStatus.

RvXMLAttrGetNameSpace()

DESCRIPTION

Returns the namespace of the *XMLAttr*.

SYNTAX

```
RvStatus RvXMLAttrGetNameSpace(
    IN    RvXMLAttrHandle    hAttr,
    INOUT RvInt32*           lenNSPrefix,
    OUT   RvChar*            strNSPrefix,
    INOUT RvInt32*           lenNS,
    OUT   RvChar*            strNS);
```

PARAMETERS

hAttr

The handle to *XMLAttr* from which to get the namespace.

lenNSPrefix

A pointer to an integer owned by the caller containing the maximum capacity of *strNSPrefix*. Upon successful completion, the integer will contain the length of the namespace prefix.

strNSPrefix

The pre-allocated buffer owned by the caller. Upon successful completion, the buffer will contain the namespace prefix.

lenNS

A pointer to an integer owned by the caller containing the maximum capacity of *strNS*. Upon successful completion, the integer will contain the length of the namespace value.

strNS

The pre-allocated buffer owned by the caller. Upon successful completion, the buffer will contain the namespace value.

RETURN VALUES

Returns RvStatus.

XML XPATH FUNCTIONS

An XML XPath processing object (*XPath*) allows addressing an element in an *XMLDoc*. This section contains the XML XPath API functions found in the *RvXMLXPath.h* header file.

According to the XPath 1.0 Specification, an expression is evaluated to yield an object, which has one of the following four basic types:

- Node-set (an unordered collection of nodes without duplicates)
- Boolean (TRUE or FALSE)
- Number (a floating-point number)
- String (a sequence of UCS characters)

A number represents a floating-point number. A number can have any double-precision 64-bit format IEEE 754 value. These include a special "Not-a-Number" (NaN) value, positive and negative infinity, and positive and negative zero.

The XML XPath API includes:

- Control Functions
- Get and Set Functions

CONTROL FUNCTIONS

The Control functions are:

- RvXMLXPathCreate()
- RvXMLXPathDestruct()
- RvXMLDocXPathEncode()
- RvXMLXPathParseQName()

RvXMLXPathCreate()

DESCRIPTION

Creates a new *XPath* processor object. An *XPath* operation is performed in the context of an *XPath* processor object. A number of *XPath* objects can be created. An *XPath* is destroyed using the RvXMLXPathDestruct() function.

SYNTAX

```
RvStatus RvXMLXPathCreate(
    IN  RvXMLMgrHandle       hMgr,
    IN  RvChar*              strXPathSource,
    OUT RvXMLXPathHandle*    phXPath);
```

PARAMETERS

hMgr

The handle to the *XMLMgr*.

strXPathSource

The *XPath* expression. The value is NULL terminated string.

phXPath

A pointer to the valid handle to an *XPath*.

RETURN VALUES

Returns RvStatus.

RvXMLXPathDestruct()

DESCRIPTION

Releases resources allocated in the previous successful calling of the RvXMLXPathCreate() and RvXMLXPathGet() functions.

SYNTAX

```
RvStatus RvXMLXPathDestruct(
    IN RvXMLXPathHandle    hXPath);
```

PARAMETERS

hXPath

The handle to the *XPath*.

RETURN VALUES

Returns RvStatus.

RvXMLDocXPathEncode()

DESCRIPTION

Encodes an *XPath* expression for the specific object in the *XMLDoc*.

SYNTAX

```
RvStatus RvXMLDocXPathEncode(
    IN    RvXMLTagHandle    hSpecificTag,
    INOUT RvInt32*          len,
    INOUT RvChar*           pDocStr);
```

PARAMETERS

hSpecificTag

The handle to the *XMLTag*.

len

The length of the string supplied by the application to which to encode the *XMLDoc*. After the call, len will contain the length of *pDocStr*.

pDocStr

The string buffer to where the *XMLDoc* will be encoded.

RETURN VALUES

Returns RvStatus.

RvXMLXPathParseQName()

DESCRIPTION

Parses the Qualified Name (QName) within a source string.

[3-6] QName ::= (Prefix ':')? LocalPart

[3-7] Prefix ::= NCName

[3-8] LocalPart ::= NCName

SYNTAX

```
RvStatus RvXMLXPathParseQName(
    IN  RvChar*     pSourceString,
    OUT RvChar**    pColonChar,
    OUT RvChar**    pEnd);
```

PARAMETERS

pSourceString

A pointer to the source string.

pColonChar

A pointer to the colon (:), if one was found.

pEnd

A pointer to the first character after the QName (may not be a whitespace).

RETURN VALUES

Returns RvStatus.

GET AND SET FUNCTIONS

The Get and Set functions are:

- RvXMLXPathSetIDAttrName()
- RvXMLXPathGet()

RvXMLXPathSetIDAttrName()

DESCRIPTION

Sets the name of the attribute which will be treated as the unique ID of the element. In particular, the *XPath* core function “id(...)” will use these attributes to select nodes.

SYNTAX

```
RvStatus RvXMLXPathSetIDAttrName(
    IN RvXMLXPathHandle    hXPath,
    IN RvChar*             strIDAttrName);
```

PARAMETERS

hXPath

The handle to the *XPath*.

strIDAttrName

The NULL-terminated string that represents the name of the attribute.

REMARKS

- By default, after the creation of an *XPath* object, the treated name is “id(...)”.
- The value of *strIDAttrName* has to be a legal attribute name. RV_XMLXPATH_IDATTR_LEN defines the maximum length of this parameter.

RETURN VALUES

Returns RvStatus.

RvXMLXPathGet()

DESCRIPTION

Parses an *XPath* expression and evaluates it.

SYNTAX

```
RvStatus RvXMLXPathGet(
    IN  RvXMLXPathHandle      hXPath,
    IN  RvXMLTagHandle        hCurrTag,
    OUT RvXMLXPathVariant*    pResult);
```

PARAMETERS

hXPath

The handle to the *XPath*.

hCurrTag

The current *XMLTag* to which the path needs to be evaluated.

pResult

A pointer to the *XPath* variant structure which will contain the result. The result type of the result depends on the *XPath* expression.

RETURN VALUES

Returns RvStatus.

REMARKS

The returned value of RV_ERROR_XPATH_NOT_MATCH means “No error during *XPath* processing, but no matching nodes”.

MEMORY POOL MODULE

This part includes the following section:

- RPOOL Functions

9

RPOOL FUNCTIONS

WHAT'S IN THIS SECTION

This section contains RPOOL functions found in the *rpool_API.h* header file. The SIP Stack does not allocate memory dynamically. All memory is allocated during the initialization process and is managed by the SIP Stack. The memory is divided to blocks called pages. The page size and the number of pages are configurable. The collection of all pages is called a memory pool. The memory pool manages the pages, supplying a simple API that allows the user to receive and recycle memory bytes when needed.

The RPOOL API enables you to construct and destruct memory pool pages, allocated pages and read and write from memory pool and allocated pages.

The RPOOL API includes:

- RPOOL Control Functions

RPOOL Control Functions

The RPOOL Control functions are:

- RPOOL_Construct()
- RPOOL_Destruct()
- RPOOL_GetPage()
- RPOOL_FreePage()
- RPOOL_CopyToExternal()
- RPOOL_AppendFromExternalToPage()
- RPOOL_Strlen()
- RPOOL_GetResources()

RPOOL_Construct()

DESCRIPTION

Allocates memory for a new pool. A pool contains a set of memory pages. The function receives parameters that indicate the number of pages and the size of each page inside the pool.

SYNTAX

```
HRPOOL RPOOL_Construct(
    IN RvInt32          pageSize,
    IN RvInt32          maxNumOfPages,
    IN RV_LOG_Handle    logHandle,
    IN RvBool           allocEmptyPages,
    IN const char*      name);
```

PARAMETERS

pageSize

The size of each page in the requested pool.

maxNumOfPages

The number of pages that should be allocated.

logHandle

LOG handle to use for log messages. You can use the SIP Stack Manager functions to get the log handle. If logHandle is NULL, no messages are printed to the log.

allocEmptyPages

Indicates whether or not the content of new allocated pages is initialized to zero.

name

Name of the RPOOL instance—used for log messages.

RETURN VALUES

Returns a handle to the RPOOL instance when the function is successful. Otherwise, the function returns NULL.

RPOOL_Destruct()

DESCRIPTION

Destructs a memory pool and frees allocated memory.

SYNTAX

```
void RPOOL_Destruct(
    IN HRPOOL    hPool);
```

PARAMETERS

hPool

The handle to the memory pool to be freed.

RETURN VALUES

None.

RPOOL_GetPage()

DESCRIPTION

Allocates a page in the memory pool.

SYNTAX

```
RvStatus RPOOL_GetPage(
    IN  HRPOOL      hPool,
    IN  RvInt32     size,
    OUT HPAGE*      newRpoolElem);
```

PARAMETERS

hPool

The handle to the RPOOL used

size

For internal use only. Use only zero.

newRpoolElem

The handle to the allocated page.

RETURN VALUES

Returns RvStatus.

RPOOL_FreePage()

DESCRIPTION

Frees a given page allocation of the memory pool.

SYNTAX

```
void RPOOL_FreePage(
    IN HRPOOL    hPool,
    IN HPAGE     hPage);
```

PARAMETERS

hPool

The handle to the RPOOL used.

hPage

The handle to the page to be deallocated.

RETURN VALUES

None.

RPOOL_CopyToExternal()

DESCRIPTION

Copies a given number of bytes from a specific location inside a given page to a specified destination buffer. This function copies non-consecutive memory into a consecutive buffer.

SYNTAX

```
RvStatus RPOOL_CopyToExternal(
    IN HRPOOL      hPool,
    IN HPAGE       hPage,
    IN RvInt32     offset,
    IN void*       dest,
    IN RvInt32     size);
```

PARAMETERS

hPool

The handle to the pool.

hPage

The handle to the source page from which the copy is made.

offset

Offset from the beginning of the page indicating from where the copy is to be made.

dest

A pointer to the destination buffer to which the copy is made.

size

The number of bytes to be copied.

RETURN VALUES

Returns RvStatus.

RPOOL_AppendFromExternalToPage()

DESCRIPTION

Copies a given number of bytes from an external buffer to the first available offset inside a given RPOOL page.

SYNTAX

```
RvStatus RPOOL_AppendFromExternalToPage(
    IN  HRPOOL          hPool,
    IN  HPAGE           hPage,
    IN  const void*     src,
    IN  int             size,
    OUT RvInt32         *allocationOffset);
```

PARAMETERS

hPool

The handle to the destination RPOOL.

hPage

The handle to the destination page.

src

A pointer to the source buffer from which the copy is made.

size

The number of bytes to be copied to the page.

allocationOffset

This is the position in the page from where the append is made. If the append fails, the output offset is UNDEFINED.

RETURN VALUES

Returns RvStatus.

RPOOL_Strlen()

DESCRIPTION

Returns the length of a NULL terminated string that is located in an RPOOL. The size of the string that this function returns does **not** include the NULL termination character. This string may reside on more the one page element.

SYNTAX

```
RvInt32 RPOOL_Strlen(
    IN HRPOOL     hPool,
    IN HPAGE      hPage,
    IN RvInt32    offset);
```

PARAMETERS

hPool

The handle to the pool.

hPage

The handle to the page.

offset

The start location of the string in the page.

RETURN VALUES

Returns the string length. (–1 is returned if the string is not a NULL terminated string).

RPOOL_GetResources()

DESCRIPTION

Returns the resources used by a given pool.

SYNTAX

```
void RPOOL_GetResources(
    IN  HRPOOL      hPool,
    OUT RvUint32    *pNumOfAlloc,
    OUT RvUint32    *pCurrNumOfUsed,
    OUT RvUint32    *pMaxUsage);
```

PARAMETERS

hPool

The handle to the pool.

pNumOfAlloc

The number of allocated blocks in the pool.

pCurrNumOfUsed

The current number of used blocks in the pool.

pMaxUsageMaxUsage

The maximum number of concurrently-used blocks in the pool until the time this function was called.

RETURN VALUES

None.

MIDDLE LAYER MODULE

The Middle Layer API functions of the Middle Layer Module interact with OS Abstraction layer (Common Core) functionality, including select/poll and timers.

This part includes the following section:

10

MIDDLE LAYER FUNCTIONS

WHAT'S IN THIS CHAPTER

This section contains Middle Layer API functions found in the *RvSipMid.h* header file. The Middle layer functions interact with OS Abstraction layer (Common Core) functionality, including select/poll and timers.

Select/poll functionality enables an application to register events for select loop or poll queries and enables an application to perform a select loop or a poll loop.

Timer functionality enables an application to set and release timers. Timers can be set on threads that run on RADVISION stacks only, since only these threads use the RADVISION select() loop.

The functions included in this section are:

MID-LAYER CONTROL FUNCTIONS

The Mid-layer Control functions are:

- RvSipMidInit()
- RvSipMidMainThreadClean() **Deprecated**
- RvSipMidEnd()
- RvSipMidConstruct()
- RvSipMidPrepareDestruct()
- RvSipMidDestruct()
- RvSipMidSetLog()
- RvSipMidMemAlloc()
- RvSipMidMemFree()
- RvSipMidTimeInMilliGet()
- RvSipMidTimeInSecondsGet()
- RvSipMidTimerSet()
- RvSipMidTimerReset()
- RvSipMidSelectCallOn()
- RvSipMidSelectSetMaxDescs()
- RvSipMidSelectGetMaxDesc()
- RvSipMidSelectGetEventsRegistration()
- RvSipMidSelectEventsHandling()
- RvSipMidPollGetEventsRegistration()
- RvSipMidPollEventsHandling()
- RvSipMidEncodeB64()
- RvSipMidDecodeB64()
- RvSipMidGetResources()
- RvSipMidTlsSetLockingCallback()
- RvSipMidTlsSetThreadIdCallback()
- RvSipMidAttachThread()
- RvSipMidDetachThread()

RvSipMidInit()

DESCRIPTION

Starts the Mid-layer services. The application must call this function before calling any of the functions in this layer.

SYNTAX

```
RvStatus RvSipMidInit(void);
```

PARAMETERS

None.

RETURN VALUES

Returns RvStatus.

RvSipMidMainThreadClean()

DESCRIPTION

This function is deprecated and kept for backward compatibility only.

SYNTAX

```
RvStatus RvSipMidMainThreadClean();
```

PARAMETERS

None.

RETURN VALUES

Returns RvStatus.

RvSipMidEnd()

DESCRIPTION

Ends the Mid-layer services.

SYNTAX

```
RvStatus RvSipMidEnd(void);
```

PARAMETERS

None.

RETURN VALUES

Returns RvStatus.

RvSipMidConstruct()

DESCRIPTION

Constructs and initializes the Mid-layer. This function allocates the required memory and constructs mid-layer objects (*mid-layers*) according to the given configuration. This function returns a handle to the Mid-layer Manager (*Mid-LayerMgr*). You need this handle in order to use the Mid-layer API functions. When you are finished using the Mid-layer, call RvSipMidPrepareDestruct() if needed, and RvSipMidDestruct() to free all allocated resources.

SYNTAX

```
RvStatus RvSipMidConstruct(
    IN  RvInt32               sizeOfCfg,
    IN  RvSipMidCfg*          pMidCfg,
    OUT RvSipMidMgrHandle*    phMidMgr);
```

PARAMETERS

pMidCfg

A structure containing Mid-layer configuration parameters.

sizeOfCfg

The size of the configuration structure.

phMidMgr

The handle to the *Mid-LayerMgr*.

RETURN VALUES

Returns RvStatus.

RvSipMidPrepareDestruct()

DESCRIPTION

Frees all application resources. After calling this function, allow enough time for other threads (if there are any) to finish calls to Mid-layer API functions and call RvSipMidDestruct(). If the application is sure that no other threads are waiting to preform actions on the Mid-layer, it can call RvSipMidDestruct() directly. After calling this function, the application is not allowed to set timers or register on select events. All application timers will be released and all select registrations will be removed.

SYNTAX

```
RvStatus RvSipMidPrepareDestruct(
    IN RvSipMidMgrHandle    hMidMgr);
```

PARAMETERS

hMidMgr

The handle to the *Mid-LayerMgr*.

RETURN VALUES

Returns RvStatus.

RvSipMidDestruct()

DESCRIPTION

Frees all *Mid-LayerMgr* resources. After calling this function, the application is not allowed to call file descriptors or set timers.

SYNTAX

```
RvStatus RvSipMidDestruct(
    IN RvSipMidMgrHandle    hMidMgr);
```

PARAMETERS

hMidMgr

The handle to the *Mid-LayerMgr*.

RETURN VALUES

Returns RvStatus.

RvSipMidSetLog()

DESCRIPTION

Sets a log handle to the *mid-layer*. Use this function if the *mid-layer w*as initiated before the SIP Stack. Use RvSipStackGetLogHandle() to get the log handle from the SIP Stack.

SYNTAX

```
RvStatus RvSipMidSetLog(
    IN RvSipMidMgrHandle    hMidMgr,
    IN RV_LOG_Handle        hLog);
```

PARAMETERS

hMidMgr

The handle to the *Mid-LayerMgr*.

hLog

The log handle.

RETURN VALUES

Returns RvStatus.

RvSipMidMemAlloc()

DESCRIPTION

Allocates memory for application use. This function can be called only after Common Core services were initialized.

SYNTAX

```
void* RvSipMidMemAlloc(
    IN RvInt32    size);
```

PARAMETERS

size

The allocation size, in bytes.

RETURN VALUES

If successful, returns a pointer to the newly allocated memory. Otherwise, returns NULL.

RvSipMidMemFree()

DESCRIPTION

Frees memory that is allocated by RvSipMidMemAlloc().

SYNTAX

```
void RvSipMidMemFree(
    IN void*    memptr);
```

PARAMETERS

memptr

The memory to free.

RETURN VALUES

None.

RvSipMidTimeInMilliGet()

DESCRIPTION

Gets a timestamp value in milliseconds.

SYNTAX

```
RvUint32 RvSipMidTimeInMilliGet(void);
```

PARAMETERS

None.

RETURN VALUES

Returns the timestamp value, in milliseconds.

RvSipMidTimeInSecondsGet()

DESCRIPTION

Gets a timestamp value in seconds.

SYNTAX

```
RvUint32 RvSipMidTimeInSecondsGet(void);
```

PARAMETERS

None.

RETURN VALUES

Returns the timestamp value, in seconds.

RvSipMidTimerSet()

DESCRIPTION

Sets a new timer. When a timer expires, the resources it consumes will be released automatically. It is forbidden to call RvSipMidTimerReset() in the timer callback.

SYNTAX

```
RvStatus RvSipMidTimerSet(
    IN  RvSipMidMgrHandle       hMidMgr,
    IN  RvUint32                miliTimeOut,
    IN  RvSipMidTimerExpEv      cb,
    IN  void*                   ctx,
    OUT RvSipMidTimerHandle*    phTimer);
```

PARAMETERS

hMidMgr

The handle to the *Mid-LayerMgr*.

miliTimeOut

The expiration time, in miliseconds.

cb

The application callback.

ctx

The context to be called when a timer expires.

phTimer

A newly allocated timer.

RETURN VALUES

Returns RvStatus.

RvSipMidTimerReset()

DESCRIPTION

Releases a timer.

SYNTAX

```
RvStatus RvSipMidTimerReset(
    IN RvSipMidMgrHandle      hMidMgr,
    IN RvSipMidTimerHandle    hTimer);
```

PARAMETERS

pMidMgr

A pointer to the *Mid-LayerMgr*.

phTimer

The handle to the timer to delete.

RETURN VALUES

Returns RvStatus.

RvSipMidSelectCallOn()

DESCRIPTION

Registers a file descriptor to the select loop. You can register to listen on read or write events, and provide a callback that will be called when the select exits due to activity on that file descriptor.

SYNTAX

```
RvStatus RvSipMidSelectCallOn(
    IN RvSipMidMgrHandle      hMidMgr,
    IN RvInt32                fd,
    IN RvSipMidSelectEvent    events,
    IN RvSipMidSelectEv       pCallBack,
    IN void*                  ctx);
```

PARAMETERS

hMidMgr

The handle to the *Mid-LayerMgr*.

fd

The OS file descriptor.

events

Read and/or write.

pCallBack

The user callback.

ctx

The user context.

RETURN VALUES

Returns RvStatus.

RvSipMidSelectSetMaxDescs()

DESCRIPTION

Sets the amount of file descriptors that the select module can handle in a single select engine. This is also the value of the highest file descriptor possible. This function **must** be called before initialization of the SIP Stack Mid-layer.

SYNTAX

```
RvStatus RvSipMidSelectSetMaxDescs(
    IN RvUint32    maxDescs);
```

PARAMETERS

maxDescs

The maximum value of file descriptor that is possible.

RETURN VALUES

Returns RvStatus.

RvSipMidSelectGetMaxDesc()

DESCRIPTION

Gets the current value used as the maximum value for a file descriptor by the select procedures.

SYNTAX

```
RvStatus RvSipMidSelectGetMaxDesc(
    OUT RvUint32    *pMaxFds);
```

PARAMETERS

pMaxFds

The place to store the maximum number of file descriptors.

RETURN VALUES

Returns RvStatus.

RvSipMidSelectGetEventsRegistration()

DESCRIPTION

Gets the select file descriptors for the select operation.

SYNTAX

```
RvStatus RvSipMidSelectGetEventsRegistration(
    IN  RvSipMidMgrHandle    hMidMgr,
    IN  RvInt                fdSetLen,
    OUT RvInt*               pMaxfd,
    OUT fd_set*              rdSet,
    OUT fd_set*              wrSet,
    OUT fd_set*              exSet,
    OUT RvUint32*            pTimeOut);
```

PARAMETERS

hMidMgr

The handle to the *Mid-LayerMgr*.

fdSetLen

The length of the file descriptors that is set.

pMaxfd

The maximum number of file descriptors.

rdSet

A pointer to the read file descriptor that was set.

wrSet

A pointer to the write file descriptor that was set.

exSet

A pointer to the exception file descriptor that was set. (Reserved for future use.)

pTimeOut

The timeout that the SIP Stack would have given to the select operation. (–1 = infinite)

RETURN VALUES

Returns RvStatus.

REMARKS

- This function can be called only in the thread that initiated the SIP Stack.
- This function resets the file descriptor sets that were given to it. If you intend to add file descriptors to the file descriptor sets, do so **after** calling this function.
- This function can be used only if the select mechanism is supported on your operating system. To verify if the select mechanism is supported, see your OS-specific Readme file.

RvSipMidSelectEventsHandling()

DESCRIPTION

Handles the events that the select procedure got.

SYNTAX

```
RvStatus RvSipMidSelectEventsHandling(
    IN RvSipMidMgrHandle    hMidMgr,
    IN fd_set*              rdSet,
    IN fd_set*              wrSet,
    IN fd_set*              exSet,
    IN RvInt                numFds,
    IN RvInt                numEvents);
```

PARAMETERS

hMidMgr

The handle to the *Mid-LayerMgr*.

rdSet

A pointer to the read file descriptor that was set.

wrSet

A pointer to write file descriptor that was set.

exSet

A pointer to the exception file descriptor that was set. (Reserved for future use.)

numFds

The maximum number of file descriptors.

numEvents

The number of occurred events.

RETURN VALUES

Returns RvStatus.

REMARKS

- This function can be called only in the thread that initiated the SIP Stack.
- This function can be used only if the select mechanism is supported on your operating system. To verify if the select mechanism is supported, see your OS-specific Readme file.

RvSipMidPollGetEventsRegistration()

DESCRIPTION

Gets the select file descriptors for the select operation.

SYNTAX

```
RvStatus RvSipMidPollGetEventsRegistration(
    IN  RvSipMidMgrHandle    hMidMgr,
    IN  RvInt                len,
    OUT struct pollfd*       pollFdSet,
    OUT RvInt*               pNum,
    OUT RvUint32*            pTimeOut);
```

PARAMETERS

hMidMgr

The handle to the *Mid-LayerMgr*.

len

The length of the poll file descriptor set.

pollFdSet

A pointer to the poll file descriptor set to poll.

pNum

The number of file descriptors on the poll set.

pTimeOut

The timeout that the SIP Stack gives to the poll. (–1= infinite)

RETURN VALUES

Returns RvStatus.

REMARKS

- This function can be called only in the thread that initiated the SIP Stack.
- This function resets the file descriptor sets that were given to it. If you intend to add file descriptors to the file descriptor sets, do so **after** calling this function.
- This function can be used only if the poll mechanism is supported on your operating system. To verify if the poll mechanism is supported, see your OS-specific Readme file.

RvSipMidPollEventsHandling()

DESCRIPTION

Handles the events that the poll procedure got.

SYNTAX

```
RvStatus RvSipMidPollEventsHandling(
    IN RvSipMidMgrHandle    hMidMgr,
    IN struct pollfd*       pollFdSet,
    IN RvInt                num,
    IN RvInt                numEvents);
```

PARAMETERS

hMidMgr

The handle to the *Mid-LayerMgr*.

num

The number of file descriptors.

pollFdSet

A pointer to the poll file descriptor that was set from the poll.

RETURN VALUES

Returns RvStatus.

REMARKS

- This function can be called only in the thread that initiated the SIP Stack.
- This function can be used only if the poll mechanism is supported on your operating system. To verify if the poll mechanism is supported, see your OS-specific Readme file.

RvSipMidEncodeB64()

DESCRIPTION

Performs a 64 bit encoding operation of a given number of bytes in a given buffer.

SYNTAX

```
RvUint32 RvSipMidEncodeB64(
    IN    RvUint8*    inTxt,
    IN    RvInt       inLen,
    INOUT RvUint8*    outTxt,
    IN    RvInt       outLen);
```

PARAMETERS

inTxt

The buffer to be encoded.

inLen

The he length of buffer to be encoded.

outTxt

The encoding destination buffer.

outLen

The size of the *outTxt* buffer.

RETURN VALUES

Returns the number of used bytes in the *outTxt* buffer or –1 if the function fails.

RvSipMidDecodeB64()

DESCRIPTION

Performs a 64 bit decoding operation of a given number of bytes in a given buffer.

SYNTAX

```
RvUint32 RvSipMidDecodeB64(
    IN    RvUint8*    inTxt,
    IN    RvInt       inLen,
    INOUT RvUint8*    outTxt,
    IN    RvInt       outLen);
```

PARAMETERS

inTxt

The buffer to be decoded.

inLen

The length of buffer to be decoded.

outTxt

The decoding destination buffer.

outLen

The size of the *outTxt* buffer.

RETURN VALUES

Returns the number of used bytes in the *outTxt* buffer or –1 if the function fails.

RvSipMidGetResources()

Gets the status of the resources that the Mid-layer uses.

SYNTAX

```
RvStatus RvSipMidGetResources(
    IN    RvSipMidMgrHandle    hMidMgr,
    INOUT RvSipMidResources    *pResources);
```

PARAMETERS

hMidMgr

The handle to the *Mid-LayerMgr*.

pResources

The resources in use by the Mid-layer.

RETURN VALUES

Returns RvStatus.

RvSipMidTlsSetLockingCallback()

DESCRIPTION

The RADVISION SIP Stack uses the OpenSSL library to provide an application with TLS capability. The OpenSSL library forces the modules that are running above it to manage locks. Specifically, OpenSSL asks the modules to be locked or unlocked on behalf of OpenSSL. This request is performed using a special callback that should be implemented in the modules, and which is called by OpenSSL. If the callback is not set into the OpenSSL library, OpenSSL does not lock shared objects, which means that multithread safety is not provided.

By default, the RADVISION SIP Stack sets the callback into the OpenSSL library on construction and removes it on destruction. See the RV_TLS_AUTO_SET_OS_CALLBACKS macro in the *rvusrconfig.h* file, which is defined as RV_YES by default. However, if the application has other modules in addition to the RADVISION SIP Stack that access the OpenSSL library, it should define the RV_TLS_AUTO_SET_OS_CALLBACKS macro as RV_NO. At this point, the application can implements its own callback, or it can use the RADVISION SIP Stack implementation. Note that the RADVISION implementation will not be available after the RADVISION SIP Stack is destructed. To activate or stop the RADVISION SIP Stack implementation of the locking callback, the RvSipMidTlsSetLockingCallback() function should be called. This function sets the callback into the OpenSSL library or removes it from the library.

SYNTAX

```
RvStatus RvSipMidTlsSetLockingCallback(
    RvBool    bSet);
```

PARAMETERS

bSet

If RV_TRUE, the callback will be set. Otherwise the callback will be removed.

RETURN VALUES

Returns RvStatus.

RvSipMidTlsSetThreadIdCallback()

DESCRIPTION

The RADVISION SIP Stack uses the OpenSSL library to provide the application with the TLS capability. To ensure multithread safety, each time OpenSSL needs to know the ID of the current thread, it asks the modules that are running above it for this ID. This request is performed using a special callback that should be implemented in the modules, and which is called by OpenSSL. If the callback is not set into the OpenSSL library, OpenSSL does not know the thread ID. As a result, multithread safety cannot be ensured.

By default, the RADVISION SIP Stack sets the callback into the OpenSSL library on construction and removes it on destruction. See the RV_TLS_AUTO_SET_OS_CALLBACKS macro in the *rvusrconfig.h* file, which is defined as RV_YES by default. However, if the application has other modules in addition to the RADVISION SIP Stack that access the OpenSSL library, it should define the RV_TLS_AUTO_SET_OS_CALLBACKS macro as RV_NO. At this point the application can implements its own callback, or it can use the RADVISION SIP Stack implementation. Note that the RADVISION implementation will not be available after the RADVISION SIP Stack is destructed. To activate or stop the RADVISION SIP Stack implementation of the Thread ID callback, the RvSipMidTlsSetThreadIdCallback() function should be called. It sets the callback into the OpenSSL library or remove it from the library.

SYNTAX

```
RvStatus RvSipMidTlsSetThreadIdCallback(
    RvBool    bSet);
```

PARAMETERS

bSet

If RV_TRUE, the callback will be set. Otherwise the callback will be removed.

RETURN VALUES

Returns RvStatus.

RvSipMidAttachThread()

DESCRIPTION

Indicates that a thread created by the application is about to use the SIP Stack API functions. This function should be called within the attached thread. Without a call to this function, calls to API functions can cause a crash. Upon thread destruction, the application must call RvSipMidDetachThread() to remove any memory that the SIP Stack allocated for this thread.

SYNTAX

```
RvStatus RvSipMidAttachThread(
    IN RvSipMidMgrHandle    hMidMgr,
    IN const RvChar         *threadName);
```

PARAMETERS

threadName

The name of this thread in the logs.

RETURN VALUES

Returns RvStatus.

RvSipMidDetachThread()

DESCRIPTION

Indicates that a thread created by the user is being shut down. This function should be called to remove any memory allocated by the SIP Stack for this thread (memory which was allocated when calling RvSipMidAttachThread()). The function should be called within the thread to detach.

SYNTAX

```
RvStatus RvSipMidDetachThread(
    IN RvSipMidMgrHandle    hMidMgr);
```

PARAMETERS

hMidMgr

The handle to the *Mid-LayerMgr*.

RETURN VALUES

Returns RvStatus.

Type Definitions and Status Codes

This part describes the type definitions and status codes of the SIP Server Platform, as included in the following sections:

- Resources Type Definitions
- SIP Server Manager Type Definitions
- Proxy Core Type Definitions
- Register Server Type Definitions
- Server Authentication Type Definitions
- Transport API Type Definitions
- Server Components Type Definitions
- SIP Server STD Type Definitions
- RPOOL Type Definitions
- Middle Layer Type Definitions
- SIP Server Common Type Definitions
- RvStatus

11

RESOURCES TYPE DEFINITIONS

WHAT'S IN THIS SECTION

This section includes the following Resources type definitions defined in the *RvSipServerResourceTypes.h* header file. The structures in this section define the resource status for each of the SIP Server modules. You should apply these structures when using the function, RvSipServerMgrGetResources().

The type definitions in this section are:

- Structures and Enumerations

STRUCTURES AND ENUMERATIONS

The type definitions are:

- RvSipServerResource
- RvSipServerMgrResources
- RvProxyPolicyResources
- RvProxyRegServerResources
- RvSipServerAuthResources
- RvSipServerDialogResources
- RvSipServerB2BResources
- RvSipServerB2BAFResources
- RvSipServerB2BAFServicesResources
- RvSipServerPresResources
- RvSipServerEventsResources
- RvSipServerWinfoResources
- RvXMLResources
- RvSipServerPublishResources
- RvSipServerPAResources
- RvSipServerLdapResources
- RvSipServerActResources

RvSipServerResource

DESCRIPTION

The declaration of the SIP Server resource structure. This object is used to receive reports about the resource consumption of one of the SIP Server modules.

SYNTAX

```
typedef RvSipResource RvSipServerResource;
```

RvSipServerMgrResources

DESCRIPTION

The resources of the SIP Server Manager module.

SYNTAX

```
typedef struct{
    RvSipServerResource    generalPoolElements;
    RvSipServerResource    smallPagesPoolElements;
    RvSipServerResource    elementLists;
    RvSipServerResource    elementListElem;
}RvSipServerMgrResources;
```

RvProxyPolicyResources

The declaration of the Proxy resource structure. This object is used to receive a report about the resource consumption of the Proxy Policy Manager module.

SYNTAX

```
typedef struct{
    RvSipServerResource    clientTransc;
    RvSipServerResource    coreObj;
    RvSipServerResource    serverTransc;
    RvSipServerResource    proxyTrx;
}RvProxyPolicyResources;
```

RvProxyRegServerResources

DESCRIPTION

The declaration of the Register Server resource structure. This object is used to receive a report about the resource consumption of the Register Server module.

SYNTAX

```
typedef struct{
    RvSipServerResource    regServerObj;
    RvSipServerResource    regServerRecord;
    RvSipServerResource    bindings;
    RvSipServerResource    recordKey;
}RvProxyRegServerResources;
```

RvSipServerAuthResources

DESCRIPTION

The declaration of the Server Authentication resource structure. This object is used to receive a report about the resource consumption of the Server Authentication module.

SYNTAX

```
typedef struct{
    RvSipServerResource    serverAuthObj;
}RvSipServerAuthResources;
```

RvSipServerDialogResources

DESCRIPTION

The declaration of the Server Dialog resource structure. This object is used to receive a report about the resource consumption of the Server Dialog module.

SYNTAX

```
typedef struct{
    RvSipServerResource    serverDialogObj;
}RvSipServerDialogResources;
```

RvSipServerB2BResources

DESCRIPTION

The declaration of the B2B resource structure. This object is used to receive a report about the resource consumption of the B2B module.

SYNTAX

```
typedef struct{
    RvSipServerResource    b2bObj;
    RvSipServerResource    b2bDlg;
    RvSipServerResource    b2bTransc;
}RvSipServerB2BResources;
```

RvSipServerB2BAFResources

DESCRIPTION

The declaration of the B2BAF resource structure. This object is used to receive a report about the resource consumption of the B2BAF module.

SYNTAX

```
typedef struct{
    RvSipServerResource    b2bafas;
    RvSipServerResource    b2bafTerm;
    RvSipServerResource    b2bafTransc;
    RvSipServerResource    routerData;
    RvSipServerResource    b2bafMasterService;
    RvSipServerResource    events;
    RvSipServerResource    lock;
    RvSipServerResource    genPool;
}RvSipServerB2BAFResources;
```

RvSipServerB2BAFServicesResources

DESCRIPTION

The declaration of the B2BAF Services resource structure. This object is used to receive a report about the resource consumption of the B2BAF Services module.

SYNTAX

```
typedef struct{
    RvSipServerResource    b2bafServices;
    RvSipServerResource    serviceConnectTranspDest;
}RvSipServerB2BAFServicesResources;
```

RvSipServerPresResources

DESCRIPTION

The declaration of the Presence resource structure. This object is used to receive a report about the resource consumption of the Presence module.

SYNTAX

```
typedef struct{
    RvSipServerResource    PresSubs;
    RvSipServerResource    PresNotification;
}RvSipServerPresResources;
```

RvSipServerEventsResources

DESCRIPTION

The declaration of the Events resource structure. This object is used to receive a report about the resource consumption of the Events module.

SYNTAX

```
typedef struct{
    RvSipServerResource    EventsSubs;
    RvSipServerResource    EventsNotify;
    RvSipServerResource    EventsHashElements;
    RvSipServerResource    EventsHashKeys;
}RvSipServerEventsResources;
```

RvSipServerWinfoResources

DESCRIPTION

The declaration of a Winfo resource structure. This object is used to receive a report about the resource consumption of the Winfo module.

SYNTAX

```
typedef struct{
    RvSipServerResource    WinfoSubs;
    RvSipServerResource    WinfoNotify;
}RvSipServerWinfoResources;
```

RvXMLResources

DESCRIPTION

The declaration of a XML resource structure. This object is used to receive a report about the resource consumption of the XML module.

SYNTAX

```
typedef struct{
    RvSipServerResource    XMLDocs;
    RvSipServerResource    XMLTags;
    RvSipServerResource    XMLAttrs;
    RvSipServerResource    XMLPages;
    RvXMLElementResource   XPathBuffers;
}RvXMLResources;
```

RvSipServerPublishResources

DESCRIPTION

The declaration of a Publish resource structure. This object is used to receive a report about the resource consumption of the Publish module.

SYNTAX

```
typedef struct{
    RvSipServerResource    PublishServer;
    RvSipServerResource    PublishHashElements;
    RvSipServerResource    PublishHashKeys;
}RvSipServerPublishResources;
```

RvSipServerPAResources

DESCRIPTION

The declaration of a Presence Agent resource structure. This object is used to receive a report about the resource consumption of the Presence Agent module.

SYNTAX

```
typedef struct{
    RvSipServerResource    PaPresentity;
    RvSipServerResource    PaPublishElements;
    RvSipServerResource    PaHashElements;
    RvSipServerResource    PaHashKeys;
    RvSipServerResource    PaXMLBuffers;
}RvSipServerPAResources;
```

RvSipServerLdapResources

DESCRIPTION

The declaration of a LDAP add-on resource structure. This object is used to receive a report about the resource consumption of the LDAP module.

SYNTAX

```
typedef struct{
    RvSipServerResource    ldapEngine;
    RvSipServerResource    ldapTransc;
}RvSipServerLdapResources;
```

RvSipServerActResources

DESCRIPTION

The declaration of ACT add-on resource structure. This object is used to receive a report about the resource consumption of the ACT module.

SYNTAX

```
typedef struct{
    RvSipServerResource    MultipartMimeParseBuffers;
    RvSipServerResource    eventQueueResources;
}RvSipServerActResources;
```

12

SIP Server Manager Type Definitions

What's in this Section

This section includes the *SipServerMgr* type definitions defined in the *RvSIPServerMgrTypes.h* header file.

The type definitions in this section are:

- Handles
- Structures and Enumerations
- Callback Functions

HANDLES

The handle is:

- RvSipServerMgrHandle
- RvSipServerAppHandle

RvSipServerMgrHandle

DESCRIPTION

The handle to the *SipServerMgr*. This handle holds information about a specific SIP Server instance.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipServerMgrHandle);
```

RvSipServerAppHandle

DESCRIPTION

The handle to the application *SipServerMgr*. This handle holds application information about a specific SIP Server instance.

SYNTAX:

```
RV_DECLARE_HANDLE(RvSipServerMgrAppHandle);
```

Structures and Enumerations

The structures and enumerations are:

- RvSipServerLogFilters
- RvSIPServerModule
- RvSipServerCfg
- RvSipServerB2BAFCfg
- RvSipServerB2BAFServiceLibCfg
- RvSipServerXMLCfg
- RvSipServerMgrEvHandler

RvSipServerLogFilters

DESCRIPTION

Defines the log filters that the SIP Server modules use. For more information, see the *SIP Server Log* chapter in the *RADVISION SIP Server Programmer Guide*.

SYNTAX

```
typedef enum{
    RVSIPSERVER_LOG_NO_LOG            =   -1,
    RVSIPSERVER_LOG_DEFAULT_FILTER    = 0x00,
    RVSIPSERVER_LOG_DEBUG_FILTER      = 0x01,
    RVSIPSERVER_LOG_INFO_FILTER       = 0x02,
    RVSIPSERVER_LOG_WARN_FILTER       = 0x04,
    RVSIPSERVER_LOG_ERROR_FILTER      = 0x08,
    RVSIPSERVER_LOG_EXCEP_FILTER      = 0x10,
    RVSIPSERVER_LOG_LOCKDBG_FILTER    = 0x20,
    RVSIPSERVER_LOG_ENTER_FILTER      = 0x40,
    RVSIPSERVER_LOG_LEAVE_FILTER      = 0x80
}RvSipServerLogFilters;
```

PARAMETERS

RVSIPSERVER_LOG_NO_LOG

No log filter will be used.

RVSIPSERVER_LOG_DEFAULT_FILTER

Uses the log filter that is defined as the default log filter for the SIP Server.

RVSIPSERVER_LOG_DEBUG_FILTER

The debug filter.

RVSIPSERVER_LOG_INFO_FILTER

The information log filter.

RVSIPSERVER_LOG_WARN_FILTER

The warning log filter.

RVSIPSERVER_LOG_ERROR_FILTER

The errors log filter.

RVSIPSERVER_LOG_EXCEP_FILTER

The exceptions log filter.

RVSIPSERVER_LOG_LOCKDBG_FILTER

The locking activities log filter.

RVSIPSERVER_LOG_ENTER_FILTER

Enters an API log filter. Only the SIP Stack uses this filter.

RVSIPSERVER_LOG_LEAVE_FILTER

Leaves an API log filter. Only the SIP Stack uses this filter.

RvSIPServerModule

DESCRIPTION

Defines the modules in the SIP Server.

SYNTAX

```
typedef enum{
       RVSIPSERVER_SERVERMGR,
       RVSIPSERVER_POLICY,
       RVSIPSERVER_TRANSCSF,
       RVSIPSERVER_STATELESS,
       RVSIPSERVER_COREAPI,
       RVSIPSERVER_SRV_AUTH_OBJ,
       RVSIPSERVER_REG_SERVER,
       RVSIPSERVER_LIST,
       RVSIPSERVER_TRANSPORT,
#ifdef RVSIPSERVER_B2B
       RVSIPSERVER_B2BUA,
#endif /*RVSIPSERVER_B2B*/
#if defined (RVSIPSERVER_B2B) || defined (RVSIPSERVER_B2BAF)
       RVSIPSERVER_DIALOG,
#endif /*(RVSIPSERVER_B2B || RVSIPSERVER_B2BAF)*/
#ifdef RVSIPSERVER_B2BAF
       RVSIPSERVER_MODULE_ID_B2BAF,
       RVSIPSERVER_MODULE_ID_B2BAF_SERVICES,
#endif /*RVSIPSERVER_B2BAF*/
#ifdef RVSIPSERVER_EVENTS
       RVSIPSERVER_PRES_SERVER,
       RVSIPSERVER_EVENTS_SERVER,
       RVSIPSERVER_WINFO,
       RVSIPSERVER_PUBLISH,
       RVSIPSERVER_PA
#endif /*RVSIPSERVER_EVENTS*/
       RVSIPSERVER_PRES_SERVER,
```

```
        RVSIPSERVER_EVENTS_SERVER,
        RVSIPSERVER_WINFO,
        RVSIPSERVER_PUBLISH,
        RVSIPSERVER_PA,
#endif /*RVSIPSERVER_EVENTS*/
        RVSIPSERVER_XML_ENCODER,
#ifdef RVSIPSERVER_LDAP
        RVSIPSERVER_LDAP_DB
#endif /*RVSIPSERVER_LDAP*/
#ifdef RVSIPSERVER_ACT
        RVSIPSERVER_ACCOUNTING,
#endif /*RVSIPSERVER_ACT*/
/*Stack modules*/
        RVSIPSERVER_RVSIP_DEFAULT,
        RVSIPSERVER_RVSIP_CORE,
        RVSIPSERVER_RVSIP_ADS,
        RVSIPSERVER_RVSIP_TRANSACTION,
        RVSIPSERVER_RVSIP_MESSAGE,
        RVSIPSERVER_RVSIP_TRANSPORT,
        RVSIPSERVER_RVSIP_PARSER,
        RVSIPSERVER_RVSIP_STACK,
        RVSIPSERVER_RVSIP_MSGBUILDER,
        RVSIPSERVER_RVSIP_AUTHENTICATOR,
        RVSIPSERVER_RVSIP_CALLLEG,
        RVSIPSERVER_RVSIP_SUBSCRIPTION,
        RVSIPSERVER_RVSIP_TRANSMITTER,
        /*Common Core modules*/
        RVSIPSERVER_RVSIP_ADS_RLIST,
        RVSIPSERVER_RVSIP_ADS_RA,
        RVSIPSERVER_RVSIP_ADS_RPOOL,
        RVSIPSERVER_RVSIP_ADS_HASH,
        RVSIPSERVER_RVSIP_ADS_PQUEUE,
        RVSIPSERVER_RVSIP_CORE_SEMAPHORE,
        RVSIPSERVER_RVSIP_CORE_MUTEX,
        RVSIPSERVER_RVSIP_CORE_LOCK,
```

```
        RVSIPSERVER_RVSIP_CORE_MEMORY,
        RVSIPSERVER_RVSIP_CORE_THREAD,
        RVSIPSERVER_RVSIP_CORE_QUEUE,
        RVSIPSERVER_RVSIP_CORE_TIMER,
        RVSIPSERVER_RVSIP_CORE_TIMESTAMP,
        RVSIPSERVER_RVSIP_CORE_CLOCK,
        RVSIPSERVER_RVSIP_CORE_TM,
        RVSIPSERVER_RVSIP_CORE_SOCKET,
        RVSIPSERVER_RVSIP_CORE_PORTRANGE,
        RVSIPSERVER_RVSIP_CORE_SELECT,
        RVSIPSERVER_RVSIP_CORE_HOST,
        RVRVSIPSERVER_RVSIP_CORE_TLS,
        RVSIPSERVER_RVSIP_CORE_ARES
}RvSipServerModule;
```

PARAMETERS

SIP SERVER MODULES

RVSIPSERVER_SIPSERVERMGR

The SIP Server Manager module.

RVSIPSERVER_POLICY

The Policy module.

RVSIPSERVER_TRANSCSF

The transaction stateful module.

RVSIPSERVER_STATELESS

The stateless module.

RVSIPSERVER_COREAPI

The Core object API.

RVSIPSERVER_SRV_AUTH_OBJ

The Server Authentication object log.

RVSIPSERVER_REG_SERVER

The Register Server module.

RVSIPSERVER_LIST

The List Handling module.

RVSIPSERVER_TRANSPORT

Server Transport module.

RVSIPSERVER_B2BUA

The B2BUA module.

RVSIPSERVER_DIALOG

The Server Dialog module.

RVSIPSERVER_MODULE_ID_B2BAF

The B2BAF module.

RVSIPSERVER_MODULE_ID_B2BAF_SERVICES

The B2BAF Services module.

RVSIPSERVER_PRES_SERVER

The Presence Server module.

RVSIPSERVER_EVENTS

The Server Events module.

RVSIPSERVER_WINFO

The Winfo module.

RVSIPSERVER_PUBLISH

The Publish Server module.

RVSIPSERVER_PA

Presence Agent Server module.

RVSIPSERVER_XML_ENCODER

The XML Encoder module.

RVSIPSERVER_LDAP_DB

The LDAP module.

RVSIPSERVER_ACCOUNTING

The Accounting module.

SIP STACK MODULES

RVSIPSERVER_RVSIP_DEFAULT

The SIP Stack default module.

RVSIPSERVER_RVSIP_ADS

The SIP Stack ADS module

RVSIPSERVER_RVSIP_CORE

The SIP Stack Core module.

RVSIPSERVER_RVSIP_TRANSACTION

The SIP Stack Transaction module.

RVSIPSERVER_RVSIP_MESSAGE

The SIP Stack Message module.

RVSIPSERVER_RVSIP_TRANSPORT

The SIP Stack Transport module.

RVSIPSERVER_RVSIP_PARSER

The SIP Stack Parser module.

RVSIPSERVER_RVSIP_STACK

The SIP Stack Manager module.

RVSIPSERVER_RVSIP_MSGBUILDER

The SIP Stack Message Builder module.

RVSIPSERVER_RVSIP_AUTHENTICATOR

The SIP Stack Authentication module.

RVSIPSERVER_RVSIP_CALLLEG

The SIP Stack Call-leg module.

RVSIPSERVER_RVSIP_SUBSCRIPTION

The SIP Stack Subscription module.

RVSIPSERVER_RVSIP_TRANSMITTER

The SIP Stack Transmitter module.

RVSIPSERVER_RVSIP_ADS_RLIST

The SIP Stack ADS RList module.

RVSIPSERVER_RVSIP_ADS_RA

The SIP Stack ADS RA module.

RVSIPSERVER_RVSIP_ADS_RPOOL

The SIP Stack ADS RPool module.

RVSIPSERVER_RVSIP_ADS_HASH

The SIP Stack ADS Hash module.

RVSIPSERVER_RVSIP_ADS_PQUEUE

The SIP Stack ADS PQueue module.

RVSIPSERVER_RVSIP_CORE_SEMAPHORE

The SIP Stack Core Semaphore module.

RVSIPSERVER_RVSIP_CORE_MUTEX

The SIP Stack Core Mutex module.

RVSIPSERVER_RVSIP_CORE_LOCK

The SIP Stack Core Lock module.

RVSIPSERVER_RVSIP_CORE_MEMORY

The SIP Stack Core Memory module.

RVSIPSERVER_RVSIP_CORE_THREAD

The SIP Stack Core Threads module.

RVSIPSERVER_RVSIP_CORE_QUEUE

The SIP Stack Core Queue module.

RVSIPSERVER_RVSIP_CORE_TIMER

The SIP Stack Core Timer module.

RVSIPSERVER_RVSIP_CORE_TIMESTAMP

The SIP Stack Core Timestamp module.

RVSIPSERVER_RVSIP_CORE_CLOCK

The SIP Stack Core Clock module.

RVSIPSERVER_RVSIP_CORE_TM

The SIP Stack Core Calendar Timer module.

RVSIPSERVER_RVSIP_CORE_SOCKET

The SIP Stack Core Socket module.

RVSIPSERVER_RVSIP_CORE_PORTRANGE

The SIP Stack Core Port-range module.

RVSIPSERVER_RVSIP_CORE_SELECT

The SIP Stack Core Select module.

RVSIPSERVER_RVSIP_CORE_HOST

The SIP Stack Core Host module.

RVSIPSERVER_RVSIP_CORE_TLS

The SIP Stack Core TLS module.

RVSIPSERVER_RVSIP_CORE_ARES

The SIP Stack Core ARES module.

RvSipServerCfg

DESCRIPTION

Contains the configuration parameters of the SIP Server Platform and the SIP Stack. When configuring to work with IPv6, you should put the addresses in square brackets []. For example, [A1B4:AB3A::AF12].

Note For a detailed description of the configuration parameters, see the *SIP Server Configuration* chapter in the *RADVISION SIP Server Platform Programmer Guide*.

SYNTAX

```
typedef struct{
/* Memory allocation */
    RvInt32            maxProxyCoreObj;
    RvInt32            maxProxyServerTransc;
    RvInt32            maxProxyClientTransc;
    RvInt32            maxServerAuthObjects;
    RvInt32            generalSipServerPoolNumOfPages;
    RvInt32            generalSipServerPoolPageSize;
    RvInt32            maxProxyElemListCount;
/* SIP Server modules log */
    RvInt32            defaultSipServerLogFilters;
    RvInt32            sipServerMgrLogFilters;
    RvInt32            policyLogFilters;
    RvInt32            transactionSFLogFilters;
    RvInt32            statelessLogFilters;
    RvInt32            coreAPILogFilters;
    RvInt32            srvAuthLogFilters;
    RvInt32            regServerLogFilters;
    RvInt32            proxyListLogFilters;
/* Application log control */
    RvSipServerPrintLogEntryEv    pfnPrintLogEntryEvHandler;
    void*              sipServerLogContext;
```

```
/* SIP Server behavior */
    RvSipServerRecordRouteMode     eRecordRoute;
    RvBool             bAuthenticateRequests;
    RvSipAuthAlgorithm eDefaultAuthAlgorithm;
    RvBool             bLoopDetectionRequired;
    RvBool             bRecurseOn3xx;
    RvBool             bAutoSend100Trying;
    RvChar*            sipServerDomainsList;
    RvInt32            sipServerDomainsListSize;
    RvSipServerMode    eSipServerMode;
    RvChar*            proxySupportedExtensionList;
    RvInt32            proxySupportedExtensionListSize;
    RvInt32            maxNumOfForwardingAttempts;
/* Registration parameters */
    RvInt32            maxRegisterNum;
    RvInt32            maxBindings;
    RvInt32            minExpireValue;
    RvInt32            maxExpireValue;
    RvInt32            defaultExpireValue;
/* Stack configuration parameters */
/* Stack memory allocation */
    RvInt32            stackMessagePoolNumOfPages;
    RvInt32            stackMessagePoolPageSize;
    RvInt32            stackGeneralPoolNumOfPages;
    RvInt32            stackGeneralPoolPageSize;
/* Stack behavior */
    RvInt32            stackRetransmissionT1;
    RvInt32            stackRetransmissionT2;
    RvInt32            stackRetransmissionT4;
    RvInt32            stackGeneralLingerTimer;
    RvInt32            stackInviteLingerTimer;
    RvInt32            stackProvisionalTimer;
    RvInt32            stackCancelGeneralNoResponseTimer;
    RvInt32            stackCancelInviteNoResponseTimer;
    RvInt32            stackGeneralRequestTimeoutTimer;
```

```
    RvInt32            stackProxy2xxRcvdTimer;
    RvInt32            stackProxy2xxSentTimer;
    RvInt32            stackMaxNumElementsInSingleDnsList;
/* Stack modules logs */
    RvInt32            stackCoreLogFilters;
    RvInt32            stackMsgLogFilters;
    RvInt32            stackTransportLogFilters;
    RvInt32            stackTransactionLogFilters;
    RvInt32            stackParserLogFilters;
    RvInt32            stackLogFilters;
    RvInt32            stackMsgBuilderLogFilters;
    RvInt32            stackAuthenticatorLogFilters;
/* Network parameters */
    RvInt32            stackSendReceiveBufferSize;
    RvChar             stackOutboundProxyIpAddress;
    RvInt32            stackOutboundProxyPort;
    RvInt32            stackMaxConnections;
    RvBool             stackTcpEnabled;
    /* Version 1.5 */
    RvInt32            viaHeaderPort;
    RvBool             bRegAuthRequests;
#ifdef RVSIPSERVER_B2B
/* Call-legs */
    RvInt32            stackCallLegLogFilters;
/* Server Dialog parameters */
    RvSipServerDialogOwner    eDialogOwner;
    RvInt32            maxStackDialogs;
    RvInt32            serverDialogLogFilters;
/* B2BUA parameters */
    RvInt32            b2bMaxB2B;
    RvInt32            b2bMaxB2BTransc;
    RvBool             b2bAuthenticationNeeded;
    RvBool             b2bAutoTry;
    RvInt32            b2bMaxDNSRetries;
    RvInt32            b2bLogFilters;
```

```
#endif /* RVSIPSERVER_B2B */
#ifdef RVSIPSERVER_EVENTS
/* General behavior */
    RvInt32            stackSubsLogFilters;
    RvBool             bSubsAuthRequests;
    RvInt32            notifyMaxDNSRetries;
    RvInt32            subsDefaultExpireValue;
/* Presence parameters */
    RvInt32            presLogFilters;
    RvInt32            presMaxNumPresSubs;
    RvInt32            presMaxNumPresNotification;
#endif /* RVSIPSERVER_EVENTS */
/* Network */
    RvChar*            localUdpAddressList;
    RvInt32            localUdpAddressListSize;
    RvChar*            localTcpAddressList;
    RvInt32            localTcpAddressListSize;
/* Multi-threading*/
    RvUint32           stackNumberOfProcessingThreads;
    RvInt32            stackProcessingQueueSize;
    RvInt32            stackNumOfReadBuffers;
/* Version 2.0 */
/* Log filters */
    RvInt32            serverTransportLogFilters;
/* Network */
    RvChar*            localTlsAddressList;
    RvInt32            localTlsAddressListSize;
/* TLS configuration */
    RvBool             bTlsEnabled;
    RvInt32            stackNumOfTlsEngines;
    RvInt32            stackMaxTlsSessions;
/* Record Route and Via lists for UDP, TCP and TLS */
    RvBool             bUseSymmetricRR;
    RvChar*            udpRecordRouteList;
    RvInt32            udpRecordRouteListSize;
```

```
    RvChar*             tcpRecordRouteList;
    RvInt32             tcpRecordRouteListSize;
    RvChar*             tlsRecordRouteList;
    RvInt32             tlsRecordRouteListSize;
    RvChar*             tlsResolvedRecordRouteList;
    RvInt32             tlsResolvedRecordRouteListSize;
    RvChar*             udpViaList;
    RvInt32             udpViaListSize;
    RvChar*             tcpViaList;
    RvInt32             tcpViaListSize;
    RvChar*             tlsViaList;
    RvInt32             tlsViaListSize;
/* Persistent connection */
    RvSipTransportPersistencyLevel  eStackPersistencyLevel;
    RvInt32             stackServerConnectionTimeout;
#ifdef RVSIPSERVER_EVENTS
/* General behavior */
    RvInt32             subsWaitingExpire;
    RvInt32             subsMinExpire;
    RvChar*             allowEventsHeader;
    RvInt32             allowEventsHeaderSize;
/* Events parameters */
    RvInt32             serverEventsLogFilters;
    RvInt32             eventsMaxNumSubs;
    RvInt32             eventsMaxNumNotify;
/* Winfo parameters */
    RvInt32             winfoLogFilters;
    RvInt32             winfoMaxNumSubs;
    RvInt32             winfoMaxNumNotify;
/*XML library */
    RvInt32             XMLMaxNumDocs;
    RvInt32             XMLMaxNumTags;
    RvInt32             XMLMaxNumAttributes;
    RvInt32             XMLMaxNumPages;
    RvInt32             XMLPageSize;
```

```
    RvInt32            XMLLogFilters;
#endif /* RVSIPSERVER_EVENTS */
#ifdef RVSIPSERVER_B2B
    RvInt32             b2bMaxB2BDlg;
#endif /* RVSIPSERVER_B2B */
    RvBool             stackUseRportParamInVia;
    RvChar*            stackOutboundProxyHostName;
    RvSipTransport     stackOutboundProxyTransport;
/* Stack parameters */
/* Report */
/* Outbound proxy host name and transport */
RvChar* stackOutboundProxyHostName;
    RvSipTransport      stackOutboundProxyTransport;
/* Version 2.5 */
#ifdef RVSIPSERVER_EVENTS
    /* Publish Parameters */
    RvInt32             pubMaxPublishServer;
    RvBool              bPubAuthRequests;
    RvInt32             pubMinExpire;
    RvInt32             pubDefaultExpire;
    RvInt32             pubNumPackages;
    RvSipServerInfoPerEventPackage*    pubInfoPerPackage;
    RvInt32             pubLogFilters;
/* Events parameters*/
    RvChar*             eventsSupportedExtensionList;
    RvInt32             eventsSupportedExtensionListSize;
    RvChar*             allowHeader;
    RvInt32             allowHeaderSize;
    RvBool              bEventsDisregardScheme;
    /* PA Parameters */
    RvInt32             paMaxPAPresentity;
    RvInt32             paMinTimer;
    RvInt32             paForceTimer;
    RvBool              bPaAddRegInfo;
    RvBool              bUsePASC;
```

```
    RvBool              bPaAggHardState;
    RvInt32             paLogFilters;
#endif /* RVSIPSERVER_EVENTS */
#ifdef RVSIPSERVER_LDAP
/* LDAP parameters */
    RvSipServerLdapServiceInfo*  pLdapServiceInfoArr;
    RvInt32               ldapNumOfServiceInfo;
    RvInt32               ldapMaxNumLdapTranscPerEngine;
    RvInt32               ldapLogFilters;
#endif /* RVSIPSERVER_LDAP */
    RvSipServerDbService  eLocationDbServiceType;
/* Session-Timer parameters */
    RvInt32               proxyMinSE;
    RvInt32               proxySessionExpires;
/* Register server supported extensions*/
    RvChar*               regServerSupportedExtensionList;
    RvInt32             regServerSupportedExtensionListSize;
#ifdef RVSIPSERVER_B2B
/* B2B parameters */
    RvChar*               b2bSupportedExtensionList;
    RvInt32               b2bSupportedExtensionListSize;
    RvInt32               b2bMinSE;
    RvInt32               b2bSessionExpires;
    RvSipSessionExpiresRefresherType   eB2BSTRefresher;
    RvBool                bB2BSupportClientSideAuth;
#endif /*RVSIPSERVER_B2B*/
    RvBool                bDisableRefer3515Behavior;
#endif /*(RVSIPSERVER_B2B || RVSIPSERVER_B2BAF)*/
/* Memory allocation */
    RvInt32               maxProxyTrx;
    RvInt32               smallPagesSize;
    RvInt32               maxSmallPages;
/* Log filters */
    RvInt32               stackTransmitterLogFilters;
    RvInt32               stackAdsRListLogFilters;
```

```
    RvInt32              stackAdsRaLogFilters;
    RvInt32              stackAdsRPoolLogFilters;
    RvInt32              stackAdsHashLogFilters;
    RvInt32              stackAdsPqueueLogFilters;
    RvInt32              stackSemaphoreLogFilters;
    RvInt32              stackMutexLogFilters;
    RvInt32              stackLockLogFilters;
    RvInt32              stackMemoryLogFilters;
    RvInt32              stackThreadLogFilters;
    RvInt32              stackQueueLogFilters;
    RvInt32              stackTimerLogFilters;
    RvInt32              stackTimeStampLogFilters;
    RvInt32              stackClockLogFilters;
    RvInt32              stackTmLogFilters;
    RvInt32              stackSocketLogFilters;
    RvInt32              stackPortRangeLogFilters;
    RvInt32              stackSelectLogFilters;
    RvInt32              stackHostLogFilters;
    RvInt32              stackTlsLogFilters;
    RvInt32              stackAresLogFilters;
    RvInt32              stackDefaultLogFilters;
    RvInt32              stackAdsLogFilters;
/* Stack memory allocation */
    RvInt32              stackElementPoolNumofPages;
    RvInt32              stackElementPoolPageSize;
/* DNS parameters */
    RvInt32              stackNumOfDnsServers;
    RvSipTransportAddr*  pStackDnsServers;
    RvInt32              stackNumOfDnsDomains;
    RvChar**             pStackDnsDomains;
    RvInt32              stackMaxDnsServers;
    RvInt32              stackMaxDnsDomains;
    RvInt32              stackMaxDnsBuffLen;
/* Version 3.0 */
    RvBool               bStackOutboundProxyEnabled;
```

```
#ifdef RVSIPSERVER_B2BAF
    RvInt32                maxGeneralCallTransc;
#endif /*RVSIPSERVER_B2BAF*/
#ifdef RVSIPSERVER_EVENTS
/* Presence parameters */
    RvBool                 bEnablePartialPresence;
    RvInt32                paXMLNumBuffers;
    RvInt32                paXMLBufferSize;
#endif /* RVSIPSERVER_EVENTS */
#ifdef RVSIPSERVER_ACT
    RvSipServerActLoggerCreatedEv    pfnActLoggerCreated;
    RvSipServerActLoggerDestroyedEv  pfnActLoggerDestroyed;
    RvSipServerActLoggerOutputInterfaceType
                                     actLoggerOutputInterface;
    RvInt32                actLoggerMaxFileSizeKb;
    RvInt32                actLoggerNumFiles;
    RvChar*                actLoggerFilenamePrefix;
    RvUint16               actRemoteLoggerListenerPort;
    RvChar*                actRemoteLoggerListenerIpAddr;
    RvInt32                actRemoteLoggerListenerIpAddrSize;
    RvChar*                actRoleOfNode;
    RvUint32               actNumActEvents;
    RvInt32                actMaxEventQueueSize;
    RvUint32               actNumActInfo;
    RvUint32               actMultiPartMimeBuffSizeByt;
    RvUint32               actNumMultiPartMimeBuff;
    RvInt32                actAcrBufferMaxSizeByt;
    RvInt32                actTriggeringFilterProxy;
    RvInt32                actTriggeringFilterRegistrar;
    RvInt32                actTriggeringFilterEvent;
    RvInt32                actTriggeringFilterPres;
    RvInt32                actTriggeringFilterWinfo;
    RvInt32                actTriggeringFilterPublish;
    RvInt32                actTriggeringFilterB2b;
    RvInt32                actTriggeringFilterServerDialog;
```

```
#endif /* RVSIPSERVER_ACT */
    RvSipServerShutdownEv pfnShutdownEvHandler;
#ifdef RVSIPSERVER_HA
/* Configuration parameters */
    RvBool                bHAEnable;
    RvInt32               iQueryHashPoolNumOfRecords;
    RvInt32               iQueryTimeout;
    RvInt32               iSocketBufferSize;
    RvChar                szBackupProcessIP;
    RvUint16              uBackupProcessPort;
    RvInt32               haLogFilters;
    RvInt32               iNumOfStoreBuffers;
#endif /* RVSIPSERVER_HA */
/*XML Manager*/
    RvSipServerXMLCfg     XMLCfg;
    RvInt32               stackConnectionCapacityPercent;
    RvChar                szUniqueServerId;
    RvInt32               maxNumOfLIR;
    RvInt32               maxNumOfDomainNames;
/* Server Dialog ST parameters */
#if defined (RVSIPSERVER_B2B) || defined (RVSIPSERVER_B2BAF)
    RvChar*          b2bafSupportedExtensionList;
    RvInt32          b2bafSupportedExtensionListSize;
    RvInt32          b2bafSTMinSE;
    RvInt32          b2bafSTSessionExpires;
    RvSipSessionExpiresRefresherType
                     b2bafSTSessionRefresher;
#endif /*(RVSIPSERVER_B2B || RVSIPSERVER_B2BAF)*/
    RvInt16                numOfElementsPerList;
    RvBool                 stackResolveTelUrls;
    RvChar*                stackDialPlanSuffix;
    RvInt32                stackRegClientLogFilters;
    RvInt32                regClientMaxRegClients;
    RvInt32                regClientAlertTimeout;
}RvSipServerCfg;
```

RvSipServerB2BAFCfg

DESCRIPTION

Contains the configuration data for the B2BAF module.

SYNTAX

```
typedef struct{
    RvInt32    sizeOfStruct;
    RvInt32    b2bafLogFilters;
    RvInt32    numOfB2BAFAS;
    RvInt32    numOfB2BAFTerm;
    RvInt32    numOfB2BAFTransc;
    RvInt32    totalNumOfServices;
    RvInt32    totalNumOfServicesCfg;
    RvInt32    numOfGeneralPages;
    RvInt32    generalPageSize;

    RvBool     bHAEnable;
    RvInt32    maxHABlobBufferSizeByte;
}RvSipServerB2BAFCfg;
```

PARAMETERS

sizeOfStruct

The size of RvSipServerB2BAFASCfg, in bytes.

b2bafLogFilters

Defines the logging filter of the B2BAF module.

Default Value: 63—full logging information.

numOfB2BAFAS

The maximum number of B2BAFAS objects (*B2BAFAS*) created by the module and existing simultaneously.

Default Value: 200

numOfB2BAFTerm

The maximum number of B2BAFTerm objects (*B2BAFTerm*) created by the module and existing simultaneously.

Default Value: 3 * numOfB2BAFAS

numOfB2BAFTransc

The maximum number of B2BAFTransc objects (*B2BAFTransc*) created by the module and existing simultaneously.

Default Value: 3 * numOfB2BAFTerm

totalNumOfServices

The maximum number of *Services* that can exist simultaneously.

Default Value: 3 * numOfB2BAFAS

totalNumOfServicesCfg

The maximum number of Services configurations that can exist simultaneously.

Default Value: 3 * totalNumOfServices

numOfGeneralPages

The maximum number of memory pages that the B2BAF and the B2BAF Services modules can allocate simultaneously. These pages are used to store received messages, addresses, and so on.

Default Value: numOfB2BAFTerm + (3 * totalNumOfServices)

generalPageSize

The size of each page (numOfGeneralPages).

Default Value: 1024

bHAEnable

Indicates whether the High Availability feature is enabled and B2BAF services are allowed to trigger HA storage.

Default Value: RV_FALSE

maxHABlobBufferSizeByte

Size of the buffer that can hold a serialized B2BAF blob.

Default Value: 4096 (4KB)

RvSipServerB2BAFServiceLibCfg

DESCRIPTION

Contains the configuration data for the B2BAF Service Library module.

SYNTAX

```
typedef struct{
    RvInt32    sizeOfStruct;
    RvInt32    numOfDestAddrPerTransparentConnectService;
    RvInt32    b2bafServiceLibLogFilters;
}RvSipServerB2BAFServiceLibCfg;
```

PARAMETERS

sizeOfStruct

The size of RvSipServerB2BAFServiceLibCfg, in bytes.

b2bafServiceLibLogFilters

Defines the logging filter of the B2BAF Services module.

Default Value: 63—full logging information.

numOfDestAddrPerTransparentConnectService

The maximum amount of destination addresses per RvSipServerB2BAFServiceTransparentConnect().

Default Value: 3

RvSipServerXMLCfg

DESCRIPTION

Contains the configuration parameters of the SIP Server XML library.

SYNTAX

```
typedef struct{
    RvUint16    sizeOfStruct;
    RvInt32     XMLMaxNumDocs;
    RvInt32     XMLMaxNumTags;
    RvInt32     XMLMaxNumAttributes;
    RvInt32     XMLMaxNumPages;
    RvInt32     XMLPageSize;
    RvInt32     XPathNumBuffers;
    RvInt32     XPathBufferSize;
    RvInt32     XMLLogFilters;
}RvSipServerXMLCfg;
```

PARAMETERS

XMLMaxNumDocs

The number of XML Document objects (*XMLDoc*) to allocate. This is the number of *XMLDoc* objects that can be allocated by the XML library simultaneously.

Default value: (2 per PAPresentity + 1 per winfoSubs + const)

XMLMaxNumTags

The number of XML Tag objects (*XMLTag*) to allocate. This is the maximum number of *XMLTag* objects for all *XMLDoc* objects handled by the XML library simultaneously.

Default value: 10 per *XMLDoc*

XMLMaxNumAttributes

The number of XML Attribute objects (*XMLAttr*) to allocate. This is the maximum number of *XMLAttr* objects for all *XMLDoc* objects handled by the XML library simultaneously.

Default value: 2 per *XMLTag*

XMLMaxNumPages

The number of memory pages to allocate for the *XMLDoc* objects. Each *XMLDoc* holds a page for all its *XMLTag* objects and *XMLAttr* objects.

Default value: 1 per *XMLDoc*

XMLPageSize

The size of memory pages to allocate for the *XMLDoc* objects.

Default value: 1024

XPathNumBuffers

The number of memory buffers to allocate for *XPath* processing. Each instance of *XPath* processor uses two buffers while processing expressions.

Default value: 2 per processing thread.

XPathBufferSize

The size of memory buffers to allocate for *XPath* processing. Longer *XPath* expressions require larger buffers for processing.

Default value: 8192

XMLLogFilters

The XML Library module log filters.

Default Value: Set by defaultLogFilters.

RvSipServerMgrEvHandler

DESCRIPTION

A structure with function pointers to the callbacks of the SIP Server. This structure is used to set the application callbacks when calling RvSipServerMgrConstructFromFile().

SYNTAX

```
typedef struct{
    RvUint16                        sizeOfStruct;
    RvSipServerPrintLogEntryEv      pfnPrintLogEntryEvHandler;
    RvSipServerShutdownEv           pfnShutdownEvHandler;
#ifdef RVSIPSERVER_ACT
    RvSipServerActLoggerCallbacks actLoggerCallbacks;
#endif /*RVSIPSERVER_ACT*/
}RvSipServerMgrEvHandler;
```

PARAMETERS

sizeOfStruct

The size of this structure. This value is needed for version control.

pfnPrintLogEntryEvHandler

Allows the user to override the default SIP Server logging and control both the log output device and the log message structure.

pfnShutdownEvHandler

Notifies the application that the SIP Server is about to shutdown.

actLoggerCallbacks

Accounting module callbacks. See the RvSipServerActLoggerCallbacks structure in the *RADVISION Accounting Add-on Module Programmer and Reference Guide* for further information.

CALLBACK FUNCTIONS

The callback function is:

- RvSipServerPrintLogEntryEv()

RvSipServerPrintLogEntryEv()

DESCRIPTION

Notifies the application each time a line should be printed to the SIP Server Log. The application can decide whether to print the line to the screen, file or other output device. You set this callback in the RvSipServerCfg structure before initializing the SIP Server. If you do not implement this function, a default logging will be used and the line will be written to the *SipLog.txt* file. Implementing this function allows you to print your messages to a different log than the default log.

SYNTAX

```
typedef void (* RvProxyPrintLogEntryEv)(
    IN void*              context,
    IN RvSipLogFilters    filter,
    IN const RvChar       *formattedText);
```

PARAMETERS

context

The context that was given in the callback registration process.

filter

The filter that this message is using, such as INFO or ERROR.

formattedText

The text to be printed to the SIP Server Log. The text is formatted as follows:

```
<filer> - <module> - <message>
```

For example:

```
INFO - SRVMGR - SIP Server was constructed successfully
```

RETURN VALUES

None.

13

Proxy Core Type Definitions

What's in this Section

This section includes the Proxy Core type definitions defined in the *RvProxyCoreTypes.h* header file.

The type definitions in this section are:

- Handles
- Structures and Enumerations
- Callback Functions

HANDLES

The handles are:

- RvProxyPolicyMgrHandle
- RvProxyPolicyMgrAppHandle
- RvProxyCoreObjHandle
- RvProxyCoreObjAppHandle
- RvProxyTranscHandle
- RvProxyTranscAppHandle
- RvProxyTrxHandle
- RvProxyTrxAppHandle

RvProxyPolicyMgrHandle

DEFINITION

The declaration of a handle to the *ProxyPolicyMgr*. The *ProxyPolicyMgr* manages all the *ProxyCoreObj* objects, the *ProxyCoreTransc* objects and the collection of *ProxyCoreTrx* objects.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyPolicyMgrHandle);
```

RvProxyPolicyMgrAppHandle

DESCRIPTION

The declaration of an application-associated handle to the *ProxyPolicyMgr*. The application uses this handle to associate the *ProxyPolicyMgr* with application-associated object. The application supplies this handle when it sets the Policy Manager Event Handler (see RvPolicyMgrEvHandler).

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyPolicyMgrAppHandle);
```

RvProxyCoreObjHandle

DESCRIPTION

The declaration of a *ProxyCoreObj* handle. This handle should be supplied in all Proxy Core Object API functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyCoreObjHandle);
```

RvProxyCoreObjAppHandle

DESCRIPTION

The declaration of an application handle to a *ProxyCoreObj*. The application uses this handle to associate a *ProxyCoreObj* with application object. The application supplies this handle when a new *ProxyCoreObj* is created The SIP Server will supply this handle to the application in each callback function related to the *ProxyCoreObj*.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyCoreObjAppHandle);
```

RvProxyTranscHandle

DESCRIPTION

The declaration of a *ProxyCoreTransc* handle. This handle is needed in all Proxy Core Transaction API functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyTranscHandle);
```

RvProxyTranscAppHandle

DESCRIPTION

The declaration of an application handle to a *ProxyCoreTransc*. The application uses this handle to associate a proxy transaction object with an application object. The application should supply this handle when a new proxy transaction object is created. The SIP Server will supply this handle back to the application in each callback function related the *ProxyCoreTransc* functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyTranscAppHandle);
```

RvProxyTrxHandle

DESCRIPTION

The declaration of a *ProxyTrx* handle. This handle is needed in all Proxy Transmitter API functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyTrxHandle);
```

RvProxyTrxAppHandle

DESCRIPTION

The declaration of application handle to a *ProxyTrx*. The application uses this handle to associate the *ProxyTrx* with the application object. The application should supply this handle when a new *ProxyTrx* is created. The proxy will supply this handle back to the application in each callback function related to the *ProxyTrx*.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyTrxAppHandle);
```

Structures and Enumerations

The structures and enumerations are:

- RvProxyCoreObjState
- RvProxyCoreObjReason
- RvProxyCoreTranscState
- RvProxyCoreTranscReason
- RvProxyCoreTrxState
- RvProxyCoreTrxReason
- RvProxyCoreForkingType
- RvProxyCoreRespAction
- RvProxyCoreObjEvHandler
- RvProxyCoreStateFulEvHandler
- RvProxyTrxStatelessEvHandler
- RvPolicyMgrEvHandler

RvProxyCoreObjState

DESCRIPTION

Represents a *ProxyCoreObj* state. The application is notified of the state of the *ProxyCoreObj* at the RvProxyCoreObjStateChangeEv() callback, where the reason for changing the state is also supplied. This enumeration defines the *ProxyCoreObj* states.

SYNTAX

```
typedef enum{
    RVPROXY_CORE_OBJ_STATE_UNDEFINED = -1,
    RVPROXY_CORE_OBJ_STATE_IDLE,
    RVPROXY_CORE_OBJ_STATE_REQUEST_RECEIVED,
    RVPROXY_CORE_OBJ_STATE_INVALID_REQUEST,
    RVPROXY_CORE_OBJ_STATE_INVALID_REQUEST_LOCAL,
    RVPROXY_CORE_OBJ_STATE_AUTHENTICATING,
    RVPROXY_CORE_OBJ_STATE_REQUEST_VALID,
    RVPROXY_CORE_OBJ_STATE_REQUEST_VALID_LOCAL,
    RVPROXY_CORE_OBJ_STATE_AUTHENTICATION_FAILED,
    RVPROXY_CORE_OBJ_STATE_RESOLVING_DESTINATION,
    RVPROXY_CORE_OBJ_STATE_ADDRESS_RESOLVED,
    RVPROXY_CORE_OBJ_STATE_DESTINATION_ADDRESS_NOT_FOUND,
    RVPROXY_CORE_OBJ_STATE_SEQ_PROXYING_REQUEST,
    RVPROXY_CORE_OBJ_STATE_PARALLEL_PROXYING_REQUEST,
    RVPROXY_CORE_OBJ_STATE_REQUEST_CANCELED,
    RVPROXY_CORE_OBJ_STATE_REQUEST_6XX_CANCELLING,
    RVPROXY_CORE_OBJ_STATE_GENERAL_FINAL_RESPONSE_SENT,
    RVPROXY_CORE_OBJ_STATE_AWAITING_ACK,
    RVPROXY_CORE_OBJ_STATE_ACK_RCVD,
    RVPROXY_CORE_OBJ_STATE_INVITE_2XX_RESPONSE_SENT,
    RVPROXY_CORE_OBJ_STATE_REQUEST_SENT,
    RVPROXY_CORE_OBJ_STATE_FINAL_RESPONSE_RCVD,
    RVPROXY_CORE_OBJ_STATE_BEFORE_TERMINATE,
    RVPROXY_CORE_OBJ_STATE_TERMINATE,
    RVPROXY_CORE_OBJ_STATE_REQUEST_CANCELLED
```

```
}RvProxyCoreObjState;
```

PARAMETERS

RVPROXY_CORE_OBJ_STATE_UNDEFINED

The state is not defined.

RVPROXY_CORE_OBJ_STATE_IDLE

The IDLE state is the initial state of the *ProxyCoreObj* state machine. When a *ProxyCoreObj* is created, the object assumes the IDLE state.

RVPROXY_CORE_OBJ_STATE_REQUEST_RECEIVED

On receipt of a request, the *ProxyCoreObj* assumes the REQUEST_RECEIVED state. The request validity is checked in this state. The application may send a provisional response or reject the request using the Proxy Core Object API functions.

RVPROXY_CORE_OBJ_STATE_INVALID_REQUEST

If the received request is Invalid, the *ProxyCoreObj* will assume the INVALID_REQUEST state. In this state, the application may validate the request by suppling a replacement, or it may reject the request.

RVPROXY_CORE_OBJ_STATE_AUTHENTICATING

After the request has been validated, if authentication is required for this *ProxyCoreObj*, the *ProxyCoreObj* assumes this state while authenticating the request.

RVPROXY_CORE_OBJ_STATE_REQUEST_VALID

If the received request is valid and was authenticated (if required for this *ProxyCoreObj* the *ProxyCoreObj* assumes the REQUEST_VALID state. In this state the application needs to call to one of the following functions in order to proceed:

- RvProxyCoreObjResolveAddr()—to proceed to the RESOLVING_DESTINATION state.
- RvProxyCoreObjSetDestAddress()—to proceed to the ADDRESS_RESOLVED state.

- RvProxyCoreObjAcceptRequest(), RvProxyCoreObjRejectRequest(), RvProxyCoreObjRedirectRequest() to proceed to one of the FINAL_RESPONSE_SENT states, according to the response class and request type.

RVPROXY_CORE_OBJ_STATE_AUTHENTICATION_FAILED

If a *ProxyCoreObj* fails, it assumes the AUTHENTICATION_FAILED state. In this state, the application can choose to reject the request or to disregard the authentication failure and proceed to RESOLVING_DESTINATION state.

RVPROXY_CORE_OBJ_STATE_RESOLVING_DESTINATION

In this state, the *ProxyCoreObj* tries to determine the destination address list (which can include one or more addresses) for the incoming request. If the resolved list is empty (meaning that no destination address was found) the *ProxyCoreObj* state is changed to DESTINATION_ADDRESS_NOT_FOUND. Otherwise, the state is changed to ADDRESS_RESOLVED.

RVPROXY_CORE_OBJ_STATE_DESTINATION_ADDRESS_NOT_FOUND

Indicates that *ProxyCoreObj* was unable to determine to which destination address the request should be forwarded. At this stage, the application can supply a set of one or more destination address to which the request should be forward, or the application can reject the request.

RVPROXY_CORE_OBJ_STATE_ADDRESS_RESOLVED

The Address Resolved state indicates that one or more destination addresses was found. In this state, the application can proxy the request or reject it.

RVPROXY_CORE_OBJ_STATE_SEQ_PROXYING_REQUEST

The application chose to proxy a request in sequential mode. Requests are sent, one at a time, to destination addresses according to the list of resolved addresses at the *ProxyCoreObj*. If a 3xx response is received, and RecurseOn3xx is on, the contacts from the 3xx response will be added to the end of the address list.

RVPROXY_CORE_OBJ_STATE_PARALLEL_PROXYING_REQUEST

The application chose to proxy a request in parallel mode. Requests are sent simultaneously to all destination addresses according to the list of resolved addresses at the *ProxyCoreObj*. If a 3xx response is received, and RecurseOn3xx is on, new requests will be sent to all addresses from the 3xx response simultaneously.

RVPROXY_CORE_OBJ_STATE_REQUEST_CANCELED

CANCEL was received for the server transaction or the application initiates Cancel(). The SIP Server is cancelling all pending client transactions and is waiting for a final response.

RVPROXY_CORE_OBJ_STATE_REQUEST_6XX_CANCELLING

A 6xx response was received (on one of the *ProxyCoreObj* branches). The SIP Server is cancelling all pending client transactions and is waiting for a final response. After all clients received a 487 response or timed out, the 6xx response will be forward as the final response.

RVPROXY_CORE_OBJ_STATE_GENERAL_FINAL_RESPONSE_SENT

The final response was forwarded or sent by the SIP Server on a non INVITE transaction.

RVPROXY_CORE_OBJ_STATE_AWAITING_ACK

A non 2xx final response was forwarded or sent by the SIP Server on an INVITE transaction. The SIP Server is waiting for ACK for this response.

RVPROXY_CORE_OBJ_STATE_ACK_RCVD

An ACK was received for the server transaction, after a non 2xx final response was sent.

RVPROXY_CORE_OBJ_STATE_INVITE_2XX_RESPONSE_SENT

A 2xx final response was forwarded on an INVITE transaction.

RVPROXY_CORE_OBJ_STATE_REQUEST_SENT

The *ProxyCoreObj* assumes this state after a request was sent and in the case that this is not a request that was forwarded by the proxy, but a request that was initiated by the proxy. For example, a CANCEL request is sent, and the request

is handled by the proxy hop to hop, meaning the proxy initiates sending a CANCEL request. In this way the *ProxyCoreObj* assumes this state after the request was sent.

RVPROXY_CORE_OBJ_STATE_FINAL_RESPONSE_RCVD

A final response was received on a request the proxy initiated. Since the proxy initiated sending the request, the response is for the proxy and the proxy does not need to forward this response.

RVPROXY_CORE_OBJ_STATE_BEFORE_TERMINATE

The last state before the final termination of the *ProxyCoreObj*. This is the last chance for the application to retrieve information from the *ProxyCoreObj*.

RVPROXY_CORE_OBJ_STATE_TERMINATE

The final state of the *ProxyCoreObj*. Upon reaching the Terminated state, the application can no longer refer to the *ProxyCoreObj*.

RVPROXY_CORE_OBJ_STATE_REQUEST_CANCELLED

When a valid CANCEL was received for a server transaction or the application initiated Cancel(), the *ProxyCoreObj* is cancels all pending client transactions and waits for a final response.

RvProxyCoreObjReason

DESCRIPTION

Describes the reason for the *ProxyCoreObj* change of state. If a request or response was invalid, this enumeration specifies the reason for rejecting the request or response.

SYNTAX

```
typedef enum{
  RVPROXY_CORE_OBJ_REASON_UNDEFINED = -1,
  RVPROXY_CORE_OBJ_REASON_ERROR,
  RVPROXY_CORE_OBJ_REASON_USER_COMMAND,
  RVPROXY_CORE_OBJ_REASON_TIME_OUT,
  RVPROXY_CORE_OBJ_REASON_NORMAL_TERMINATION,
  RVPROXY_CORE_OBJ_REASON_REQUEST_RCVD,
  RVPROXY_CORE_OBJ_REASON_REQUEST_VALID,
  RVPROXY_CORE_OBJ_REASON_REQUEST_LOOP_DETECTED,
  RVPROXY_CORE_OBJ_REASON_REQUEST_ILLEGAL_MAX_FORWARD,
  RVPROXY_CORE_OBJ_REASON_REQUEST_UNSUPPORTED_EXTENSION,
  RVPROXY_CORE_OBJ_REASON_REQUEST_UNKNOWN_SCHEME,
  RVPROXY_CORE_OBJ_REASON_REQUEST_ADDR_FOUND,
  RVPROXY_CORE_OBJ_REASON_REQUEST_ADDR_NOT_FOUND,
  RVPROXY_CORE_OBJ_REASON_REQUEST_AUTH_SUCCEED,
  RVPROXY_CORE_OBJ_REASON_REQUEST_AUTH_FAIL,
  RVPROXY_CORE_OBJ_REASON_REQUEST_AUTH_ERROR,
  RVPROXY_CORE_OBJ_REASON_CANCEL_RCVD,
  RVPROXY_CORE_OBJ_REASON_RESPONSE_VALID,
  RVPROXY_CORE_OBJ_REASON_FORWARD_REQUEST,
  RVPROXY_CORE_OBJ_REASON_INITIATE_REQUEST,
  RVPROXY_CORE_OBJ_REASON_FORWARD_RESPONSE,
  RVPROXY_CORE_OBJ_REASON_NETWORK_ERROR,
  RVPROXY_CORE_OBJ_REASON_RESPONSE_DISCARD,
  RVPROXY_CORE_OBJ_REASON_INVALID_RESPONSE,
  RVPROXY_CORE_OBJ_REASON_RESPONSE_ILLEGAL_ACTION,
  RVPROXY_CORE_OBJ_REASON_MSG_SENT,
```

```
  RVPROXY_CORE_OBJ_REASON_MSG_RCVD,
  RVPROXY_CORE_OBJ_REASON_STACK_TRANSC_TERMINATED,
  RVPROXY_CORE_OBJ_REASON_REG_SERVER_OBJ_TERMINATED,
#ifdef RVSIPSERVER_EVENTS
  RVPROXY_CORE_OBJ_REASON_PUBLISH_SERVER_TERMINATED
#endif /*RVSIPSERVER_EVENTS*/
  RVPROXY_CORE_OBJ_REASON_BAD_SYNTAX,
  RVPROXY_CORE_OBJ_REASON_REQUEST_SESSION_EXPIRES_TOO_SMALL,
  RVPROXY_CORE_OBJ_REASON_TRANSAC_TIMEOUT,
  RVPROXY_CORE_OBJ_REASON_NETWORK_ERROR,
  RVPROXY_CORE_OBJ_REASON_503_FAILURE_RECVD
}RvProxyCoreObjReason;



typedef enum{
    RVPROXY_CORE_OBJ_REASON_UNDEFINED = -1,
    RVPROXY_CORE_OBJ_REASON_ERROR,
    RVPROXY_CORE_OBJ_REASON_USER_COMMAND,
    RVPROXY_CORE_OBJ_REASON_NORMAL_TERMINATION,
    RVPROXY_CORE_OBJ_REASON_REQUEST_RCVD,
    RVPROXY_CORE_OBJ_REASON_REQUEST_VALID,
    RVPROXY_CORE_OBJ_REASON_REQUEST_LOOP_DETECTED,
    RVPROXY_CORE_OBJ_REASON_REQUEST_ILLEGAL_MAX_FORWARD,
    RVPROXY_CORE_OBJ_REASON_REQUEST_UNSUPPORTED_EXTENSION,
    RVPROXY_CORE_OBJ_REASON_REQUEST_UNKNOWN_SCHEME,
    RVPROXY_CORE_OBJ_REASON_REQUEST_ADDR_FOUND,
    RVPROXY_CORE_OBJ_REASON_REQUEST_ADDR_NOT_FOUND,
    RVPROXY_CORE_OBJ_REASON_REQUEST_AUTH_SUCCEED,
    RVPROXY_CORE_OBJ_REASON_REQUEST_AUTH_FAIL,
    RVPROXY_CORE_OBJ_REASON_REQUEST_AUTH_ERROR,
    RVPROXY_CORE_OBJ_REASON_CANCEL_RCVD,
    RVPROXY_CORE_OBJ_REASON_RESPONSE_VALID,
    RVPROXY_CORE_OBJ_REASON_FORWARD_REQUEST,
    RVPROXY_CORE_OBJ_REASON_INITIATE_REQUEST,
```

```
    RVPROXY_CORE_OBJ_REASON_FORWARD_RESPONSE,
    RVPROXY_CORE_OBJ_REASON_RESPONSE_DISCARD,
    RVPROXY_CORE_OBJ_REASON_INVALID_RESPONSE,
    RVPROXY_CORE_OBJ_REASON_RESPONSE_ILLEGAL_ACTION,
    RVPROXY_CORE_OBJ_REASON_MSG_SENT,
    RVPROXY_CORE_OBJ_REASON_MSG_RCVD,
    RVPROXY_CORE_OBJ_REASON_STACK_TRANSC_TERMINATED,
    RVPROXY_CORE_OBJ_REASON_REG_SERVER_OBJ_TERMINATED,
#ifdef RVSIPSERVER_EVENTS
    RVPROXY_CORE_OBJ_REASON_PUBLISH_SERVER_TERMINATED,
#endif /*#ifdef RVSIPSERVER_EVENTS*/
    RVPROXY_CORE_OBJ_REASON_BAD_SYNTAX,
RVPROXY_CORE_OBJ_REASON_REQUEST_SESSION_EXPIRES_TOO_SMALL,
    RVPROXY_CORE_OBJ_REASON_TRANSAC_TIMEOUT,
    RVPROXY_CORE_OBJ_REASON_NETWORK_ERROR,
    RVPROXY_CORE_OBJ_REASON_503_FAILURE_RECVD
}RvProxyCoreObjReason;
```

RvProxyCoreTranscState

DESCRIPTION

Represents a *ProxyCoreTransc* state. The application is notified of the state of the *ProxyCoreTransc* at the RvProxyCoreTranscStateChangeEv callback, together with the reason for changing the state.

SYNTAX

```
typedef enum{
    RVPROXY_CORE_TRANSC_STATE_UNDEFINED = -1,
    RVPROXY_CORE_TRANSC_STATE_IDLE,
    RVPROXY_CORE_TRANSC_STATE_REQUEST_SENT,
    RVPROXY_CORE_TRANSC_STATE_MSG_SEND_FAILURE,
    RVPROXY_CORE_TRANSC_STATE_BEFORE_TERMINATED,
    RVPROXY_CORE_TRANSC_STATE_TERMINATED
}RvProxyCoreTranscState;
```

PARAMETERS

RVPROXY_CORE_TRANSC_STATE_UNDEFINED

An undefined state.

RVPROXY_CORE_TRANSC_STATE_IDLE

The state in which the transaction is created. After sending a request, the *ProxyCoreTransc* changes to the RVPROXY_CORE_TRANSC_STATE_REQUEST_SENT state.

RVPROXY_CORE_TRANSC_STATE_REQUEST_SENT

The *ProxyCoreTransc* assumes this state after sending an outgoing request.

RVPROXY_CORE_MSG_TRANSC_STATE_SEND_FAILURE

Indicates that sending a message (request) failed. In the current release, relevant only for the sending of requests.

Upon assuming this state, the application can choose whether to continue sending the request using RvProxyCoreTranscDnsContinue(), or terminate the current *ProxyCoreTransc* using RvProxyCoreTranscDnsTerminate(). The *ProxyCoreTransc* stays in this state until the application chooses whether to terminate or continue. If the application chooses to continue, the *ProxyCoreTransc* assumes the RVPROXY_CORE_TRANSC_STATE_REQUEST_SENT state.

RVPROXY_CORE_TRANSC_STATE_BEFORE_TERMINATED

The last state before the final termination of the*ProxyCoreTransc*. This is the last chance for the application to retrieve information from the *ProxyCoreTransc*.

RVPROXY_CORE_TRANSC_STATE_TERMINATED

The final state of the *ProxyCoreTransc*.Upon reaching the Terminated state, the application can no longer refer to the *ProxyCoreTransc*.

RvProxyCoreTranscReason

DESCRIPTION

Describes the reason for the *ProxyCoreTransc* validation failure or termination.

SYNTAX

```
typedef enum{
  RVPROXY_CORE_TRANSC_REASON_UNDEFINED = -1,
  RVPROXY_CORE_TRANSC_REASON_NO_VIA,
  RVPROXY_CORE_TRANSC_REASON_TOP_VIA_NOT_MATCH,
  RVPROXY_CORE_TRANSC_REASON_NO_VIA_LEFT,
  RVPROXY_CORE_TRANSC_REASON_TRANSAC_TIMEOUT,
  RVPROXY_CORE_TRANSC_REASON_ERROR,
  RVPROXY_CORE_TRANSC_REASON_NETWORK_ERROR,
  RVPROXY_CORE_TRANSC_REASON_RESPONSE_503_FAILURE_RECVD,
  RVPROXY_CORE_TRANSC_REASON_USER_COMMAND,
  RVPROXY_CORE_TRANSC_REASON_SENDING_REQUEST,
  RVPROXY_CORE_TRANSC_REASON_REQUEST_CANCELLED,
  RVPROXY_CORE_TRANSC_REASON_TERMINATE_NORMAL,
  RVPROXY_CORE_TRANSC_REASON_TERMINATE_CORE_OBJ_TERMINATED,
  RVPROXY_CORE_TRANSC_REASON_TERMINATE_ERROR,
  RVPROXY_CORE_TRANSC_REASON_PROCEEDING_TIMEOUT,
  RVPROXY_CORE_TRANSC_REASON_BAD_SYNTAX
}RvProxyCoreTranscReason;
```

RvProxyCoreTrxState

DESCRIPTION

Represents the state of a *ProxyTrx*. The application is notified of the *ProxyTrx* state at the RvProxyCoreTrxStateChangeEv() callback, with the reason for the state change.

SYNTAX

```
typedef enum{
    RVPROXY_CORE_TRX_STATE_UNDEFINED = -1,
    RVPROXY_CORE_TRX_STATE_IDLE,
    RVPROXY_CORE_TRX_STATE_RESOLVING_ADDR,
    RVPROXY_CORE_TRX_STATE_FINAL_DEST_RESOLVED,
    RVPROXY_CORE_TRX_STATE_READY_FOR_SENDING,
    RVPROXY_CORE_TRX_STATE_MSG_SEND_FAILURE,
    RVPROXY_CORE_TRX_STATE_MSG_SENT,
    RVPROXY_CORE_TRX_STATE_TERMINATED
}RvProxyCoreTrxState;
```

PARAMETERS

RVPROXY_CORE_TRX_STATE_UNDEFINED

An undefined state.

RVPROXY_CORE_TRX_STATE_IDLE

The state in which the *ProxyTrx* is created. After RvProxyCoreTrxSendMsg() is called, the *ProxyTrx* changes its state to RVPROXY_TRX_STATE_RESOLVING_ADDR.

RVPROXY_CORE_TRX_STATE_RESOLVING_ADDR

Indicates that the *ProxyTrx* is about to start the address resolution process.

RVPROXY_CORE_TRX_STATE_READY_FOR_SENDING

Indicates that the *ProxyTrx* is ready to send the message to the remote party.

RVPROXY_CORE_TRX_STATE_MSG_SEND_FAILURE

Indicates that sending a message failed. Upon assuming this state, the application can choose whether to continue sending the request using RvProxyCoreTrxSendMsg(), or terminate the current *ProxyTrx* using RvProxyCoreTrxTerminate(). The *ProxyTrx* stays in this state until the application chooses whether to terminate or continue. If the application chooses to continue, the *ProxyTrx* will assume the RVPROXY_TRX_STATE_MSG_SENT state.

RVPROXY_CORE_TRX_STATE_MSG_SENT

The *ProxyTrx* assumes this state after successfully sending an outgoing message.

RVPROXY_CORE_TRX_STATE_TERMINATED

This is the final state of the *ProxyTrx*. Upon reaching this state, the application can no longer refer to the *ProxyTrx*.

RvProxyCoreTrxReason

DESCRIPTION

Describes the reason for the *ProxyTrx* validation failure or the state change reason.

SYNTAX

```
typedef enum{
    RVPROXY_CORE_TRX_REASON_UNDEFINED = -1,
    RVPROXY_CORE_TRX_REASON_ERROR,
    RVPROXY_CORE_TRX_REASON_NETWORK_ERROR,
    RVPROXY_CORE_TRX_REASON_CONNECTION_ERROR,
    RVPROXY_CORE_TRX_REASON_OUT_OF_RESOURCES,
    RVPROXY_CORE_TRX_REASON_TERMINATE_NORMAL,
    RVPROXY_CORE_TRX_REASON_CORE_OBJ_TERMINATED,
    RVPROXY_CORE_TRX_REASON_FINAL_ADDR_FOUND,
    RVPROXY_CORE_TRX_REASON_MSG_SENT,
    RVPROXY_CORE_TRX_REASON_USER_COMMAND
}RvProxyCoreTrxReason;
```

RvProxyCoreForkingType

DESCRIPTION

The forwarding behavior of the requests—whether the address will be handled one by one or all at once.

SYNTAX

```
typedef enum{
    RVPROXY_FORKING_UNDEFINED = -1,
    RVPROXY_SEQ_FORKING,
    RVPROXY_PARALLEL_FORKING
}RvProxyCoreForkingType;
```

PARAMETERS

RVPROXY_FORKING_UNDEFINED

An invalid forking type.

RVPROXY_SEQ_FORKING

Forwards the request with sequential forking. The request will be sequentially sent to each of the address in the destinations addresses list, until a 2xx response is received, or the list is empty.

If a 3xx response is received, and RecurseOn3xx is on, the contacts will be added to the end of the decantations addresses list.

RVPROXY_PARALLEL_FORKING

Forwards a request with parallel forking. The request will be sent in parallel to all addresses in the destinations addresses list.

If a 3xx response is received, and RecurseOn3xx is on, new requests will be sent at once to all addresses according to the contacts in the 3xx response.

RvProxyCoreRespAction

The action to be done on a received response by the SIP Server according to the RFC and the SIP Server internal logic.

SYNTAX

```
typedef enum{
    RVPROXY_RESPONSE_UNDEFINED = -1,
    RVPROXY_RESPONSE_FORWARD,
    RVPROXY_RESPONSE_STORE,
    RVPROXY_RESPONSE_DISCARD,
    RVPROXY_RESPONSE_RECURSE,
    RVPROXY_RESPONSE_WAIT_APP
}RvProxyCoreRespAction;
```

PARAMETERS

RVPROXY_RESPONSE_UNDEFINED

An undefined action.

RVPROXY_RESPONSE_FORWARD

The proxy is going to forward the response received.

RVPROXY_RESPONSE_STORE

The proxy will store the received response. 3xx-5xx responses should not be forward immediately, but stored in the response context. If no response was forwarded and all client transactions terminated, the proxy should choose the best response from all stored responses.

In sequential proxying, after storing the response, continues to the next address in the resolved address list.

RVPROXY_RESPONSE_DISCARD

The proxy will discard the message. This may happen if a final response was already forwarded, and a new response (which is not 2xx) is received, or if a 1xx response is received after a final response was forwarded.

RVPROXY_RESPONSE_RECURSE

The proxy will add the address/addresses received in the 3xx response to the end of the address list (in sequential forking), or send new requests to all addresses at once (in parallel proxying).

Relevant only in the case that the RecurseOn3xx flag is on.

RVPROXY_RESPONSE_WAIT_APP

According to the internal logic of the SIP Server, the response should be discarded and the SIP Server will not continue with proxying the request until the application decides how to continue. This response action is recommended by the SIP Server when a 503 response was received when in sequential forking and working with DNS. In this case, the application must decide whether to terminate the transaction (and a 503 will be forwarded) or to continue to the next address in the DNS address list.

RvProxyCoreObjEvHandler

DESCRIPTION

A structure with function pointers to the module callbacks, relevant when working in stateful or stateless mode. This structure is used to set the application callback implementations using RvProxyCorePolicyMgrSetCoreObjEvHandler().

SYNTAX

```
typedef struct{
    RvProxyCoreObjectCreatedEv
                            pfnEvCoreObjCreated;
    RvProxyCoreObjStateChangeEv
                            pfnEvCoreObjStateChange;
    RvProxyCoreObjMsgReceivedEv
                            pfnEvCoreObjMsgRcvd;
    RvProxyCoreObjMsgToSendEv
                            pfnEvCoreObjMsgToSend;
    RvProxyCoreObjProcessResponseEv
                            pfnEvCoreObjProcessResponse;
    RvProxyCoreObjInitiateResponseEv
                            pfnEvCoreObjInitiateResponse;
    RvProxyCoreObjInvalidResponseEv
                            pfnEvCoreObjInvalidResponse;
    RvProxyCoreObjSupplyRecordRouteForReqEv
                            pfnEvCoreObjSupplyRRParamsForReq;
    RvProxyCoreObjRewriteRecordRouteInResp
                            pfnEvCoreObjRewriteRRInResp;
    RvProxyCoreObjRequestFinalDestResolvedEv
                            pfnEvCoreObjFinalDestResolved;
    RvProxyCoreObjSLRequestFinalDestResolvedEv
                            pfnEvCoreObjSLFinalDestResolved;
    RvProxyCoreObjOtherURLRcvdEv
                            pfnEvCoreObjOtherURLRcvd;
}RvProxyCoreObjEvHandler;
```

PARAMETERS

pfnEvCoreObjCreated

Notifies of the creation of a *ProxyCoreObj*.

pfnEvCoreObjNewRequestRcvd

Called when a new request is received and the application had configured the Policy Manager to work in dynamic mode (RVSIPSERVER_MODE_DYNAMIC). The application should decide whether to handle this request in transaction stateful or stateless modes.

pfnEvCoreObjStateChange

Notifies of a *ProxyCoreObj* state change.

pfnEvCoreObjMsgRcvd

Notifies that a message was received.

pfnEvCoreObjMsgToSend

Notifies that a message is about to be sent.

pfnEvCoreObjProcessResponse

Notifies how the proxy is going to handle an incoming response.

pfnEvCoreObjInitiateResponse

Notifies that the proxy is about to initiate a response message.

pfnEvCoreObjInvalidResponse

Notifies the application about an incoming invalid response. The response is discarded, and the *ProxyCoreObj* may be terminated (according to the *ProxyCoreObj* state).

pfnEvCoreObjSupplyRRParamsForReq

Allows the application to set the Record Route header of the request to be sent.

pfnEvCoreObjRewriteRRInResp

Allows the application to rewrite the Record Route header while forwarding a response.

pfnEvCoreObjFinalDestResolved

Notifies the application that the SIP Server "knows" the explicit IP address port and transport that is going to send the specified stateful request message. Before the callback is called the SIP Server sets the local interface it will use to send the request. Choosing the address will be based on the first match according to address type, transport and port. On this event, the application can call RvProxyCoreTranscGetResolvedFinalDest() to retrieve this information. Using this information the application can base it decision on which local interface will be used to forward this request.

pfnEvCoreObjSLFinalDestResolved

Notifies the application that the SIP Server "knows" the explicit IP address port and transport that is going to send the specified stateless request message. Before the callback is called, the SIP Server sets the local interface it will use to send the request. Choosing the address will be based on the first match according to address type, transport and port. On this event, the application retrieves the final destination of the request (address, transport and port). Using this information, the application can base its decision on which local interface will be used to forward this request. The application must return these values (address, port and transport) in the OUT parameters.

Note You must not lock your *hAppPolicyMgr* object on this event to avoid deadlock.

RvProxyCoreStateFulEvHandler

DESCRIPTION

A structure with function pointers to the callbacks of the module. This structure is used to set the application callbacks that correspond with *ProxyCoreObj* objects. To set these callbacks, the application needs to call to RvProxyCorePolicyMgrSetStatefulEvHandler().

SYNTAX

```
typedef struct{
    RvProxyCoreTranscCreatedEv
                              pfnEvCoreTransacCreated;
    RvProxyCoreTranscStateChangeEv
                              pfnEvCoreTranscStateChange;
    RvProxyCoreObjectInternalCreatedEv
                              pfnEvCoreObjInternalCreated;
    RvProxyCoreObjBestRespToForwardEv
                              pfnEvCoreObjBestRespToForward;
    RvProxyCoreTranscOtherURLToSendEv
                              pfnEvCoreTranscOtherURLToSend;
    RvProxyCoreTranscNewConnInUseEv
                              pfnEvCoreTranscNewConnInUse;
    }RvProxyCoreStateFulEvHandler;
```

PARAMETERS

pfnEvCoreTransacCreated

Notifies of the creation of a client transaction.

pfnEvCoreTranscStateChange

Notifies of a client transaction state change.

pfnEvCoreObjInternalCreated

Notifies that a internal core object has been created as a reaction to Cancel().

pfnEvCoreObjBestRespToForward

Notifies about the message chosen as best response and enables the application to set a different message as best response.

pfnEvCoreTranscOtherURLToSend

Notifies the application that a message needs to be sent (in stateful mode), and the destination address is a URL type that the proxy does not currently support. In order for the message to be sent, the application must convert this URL into a SIP URL (supply the result in hSipURLAddress).

pfnEvCoreTranscNewConnInUse

Notifies that the transaction is now using a new connection.

RvProxyTrxStatelessEvHandler

DESCRIPTION

A structure with function pointers to the callbacks of the module. This structure is used to set the application callbacks that correspond with *ProxyTrx* objects. To set these callback, the application needs to call RvProxyCorePolicyMgrSetStatelessEvHandler().

SYNTAX

```
typedef struct{
    RvProxyCoreTrxCreatedEv
                                pfnEvProxyTrxCreated;
    RvProxyCoreTrxStateChangeEv
                                pfnEvProxyTrxStateChange;
    RvProxyCoreTrxOtherURLToSendEv
                                pfnEvProxyTrxOtherURLToSend;
}RvProxyTrxStatelessEvHandler;
```

PARAMETERS

pfnEvProxyTrxCreated

Notifies the application of the creation of a *ProxyTrx*.

pfnEvProxyTrxStateChange

Notifies the application of state changes of the *ProxyTrx*.

pfnEvProxyTrxOtherURLToSend

Notifies the application about a message that needs to be sent whose destination address is a URL type that the SIP Stack does not currently support. The application must convert the URL to a SIP URL for the message to be sent.

RvPolicyMgrEvHandler

DESCRIPTION

Contains events that are relevant to deciding how to handle an incoming request—as a Proxy Server, Registrar, Presence, or B2BUA.

SYNTAX

```
typedef struct{
    RvProxyPolicyMgrNewRequestRcvdEv
                          pfnEvPolicyMgrNewRequestRcvd;
    RvProxyPolicyMgrHandleLocalRegEv
                          pfnEvPolicyMgrHandleLocalReg;
    #ifdef RVSIPSERVER_EVENTS
    RvProxyPolicyMgrHandleLocalSubsEv
                          pfnEvPolicyMgrHandleLocalSubs;
    #endif /*RVSIPSERVER_EVENTS*/
    vProxyPolicyMgrOpenDialogEv
                          pfnEvPolicyMgrOpenDialog;
    #ifdef RVSIPSERVER_EVENTS
    RvProxyPolicyMgrHandleLocalPublishEv
                          pfnEvPolicyMgrHandleLocalPublish;
    #endif /*RVSIPSERVER_EVENTS*/
          RvProxyPolicyMgrOtherURLRcvdEv
                          pfnEvPolicyMgrOtherURLRcvd;
    }RvPolicyMgrEvHandler;
```

PARAMETERS

pfnEvPolicyMgrNewRequestRcvd

A new request was received. Consults the application whether to handle the request as stateless or stateful.

pfnEvPolicyMgrHandleLocalReg

A REGISTER request was received with a Request URI of a domain for which the server is responsible. Consults the application whether to handle this request locally, as a Registrar, or forward it using the Proxy Server.

pfnEvPolicyMgrHandleLocalSubs

Consults the application whether to handle an incoming SUBSCRIBE request locally by the local Presence Server, or whether the Subscribe should be forwarded and then handled by the Proxy Server. This event will be called when a new SUBSCRIBE request is received for the local domain.

pfnEvPolicyMgrOpenDialog

A new initial request that can initiate a dialog was received. Consults the application whether to open a dialog or handle the request using the Proxy Server.

pfnEvPolicyMgrHandleLocalPublish

Consults the application on whether the local Presence Server should handle an incoming PUBLISH request, or whether the PUBLISH request should be forwarded and then handled by the Proxy Server.

pfnEvPolicyMgrOtherURLRcvd

Consults the application on whether or not an address whose scheme is not supported by the SIP Stack should be considered as one of the local domains of the proxy.

CALLBACK FUNCTIONS

The callback functions include:

- Proxy Policy Manager Events
- Proxy Core Object Events
- Stateful Proxy Core Object Events
- Stateless Proxy Core Object Events

Proxy Policy Manager Events

The Proxy Policy Manager events are:

- RvProxyPolicyMgrNewRequestRcvdEv()
- RvProxyPolicyMgrHandleLocalRegEv()
- RvProxyPolicyMgrHandleLocalSubsEv()
- RvProxyPolicyMgrOpenDialogEv()
- RvProxyPolicyMgrHandleLocalPublishEv()
- RvProxyPolicyMgrOtherURLRcvdEv()

RvProxyPolicyMgrNewRequestRcvdEv()

DESCRIPTION

Called when a new request is received and the application configures the *ProxyPolicyMgr* to work in Dynamic mode (RVSIPSERVER_MODE_DYNAMIC, for example). The application should decide whether to handle this request in transaction stateful or stateless mode.

SYNTAX

```
typedef void (* RvProxyPolicyMgrNewRequestRcvdEv)(
    IN  RvProxyPolicyMgrHandle       hPolicyMgr,
    IN  RvProxyPolicyMgrAppHandle    hAppObj,
    IN  RvSipMsgHandle               hRcvdRequest,
    OUT RvBool                       *pHandleStatefully);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

hAppObj

The application handle for this *ProxyPolicyMgr*.

hRcvdRequest

The received request.

pHandleStatefully

The answer returned by the application. RV_TRUE if the application handle the request using the transaction stateful layer. Otherwise, RV_FALSE.

RETURN VALUES

None.

RvProxyPolicyMgrHandleLocalRegEv()

DESCRIPTION

Notifies the application that the current Register request is a local request. The application should decide how to handle this Registration request—as a Local Registration, using the *RegServerObj* and its API functions or handle the request as all other incoming requests. In this case, the Proxy Core will handle the request as a general request.

SYNTAX

```
typedef void (* RvProxyPolicyMgrHandleLocalRegEv)(
    IN  RvProxyPolicyMgrHandle       hPolicyMgr,
    IN  RvProxyPolicyMgrAppHandle    hAppMgr,
    IN  RvSipMsgHandle               hRegRequest,
    OUT RvBool*                      pbHandleLocal);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

hAppMgr

The handle to the application object of the *ProxyPolicyMgr*.

hRegRequest

The Register request that should be handled.

pbHandleLocalReg

RV_TRUE if the application handles the request as a local registration using the *RegisterServer* API. RV_FALSE if the application handles the request as a general request using the *ProxyCoreObj* API.

RETURN VALUES

None.

RvProxyPolicyMgrHandleLocalSubsEv()

DESCRIPTION

Consults with the application on whether to handle an incoming SUBSCRIBE request locally, by the local Presence Server, or whether the SUBSCRIBE should be forwarded and then handled by the Proxy Server. This event is called when a new SUBSCRIBE request is received and the Request URI is for the local domain.

Note This function is relevant only if you have acquired the RADVISION SIP Events Server Add-on Module.

SYNTAX

```
typedef void (* RvProxyPolicyMgrHandleLocalSubsEv)(
    IN  RvProxyPolicyMgrHandle       hPolicyMgr,
    IN  RvProxyPolicyMgrAppHandle    hAppMgr,
    IN  RvSipMsgHandle               hSubsRequest,
    OUT RvBool*                      pbHandleLocal);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

hAppMgr

The handle of the application *ProxyPolicyMgr*.

hSubsRequest

The handle of the received SUBSCRIBE request.

pbHandleLocal

The returned value according to the decision of the application.

RV_TRUE if the application handles the request locally using the Presence Server. RV_FALSE if the application handles the request using the Proxy Server as with all other requests. (The request can be forwarded or rejected).

RETURN VALUES

None.

RvProxyPolicyMgrOpenDialogEv()

DESCRIPTION

Consults with the application on whether to open a Dialog for handling the request or to handle the request as a Proxy Server. This event is called when a new request that can initiate a Dialog is received (currently, only for initial INVITE requests) and the application is working in Dynamic mode.

SYNTAX

```
typedef void (* RvProxyPolicyMgrOpenDialogEv)(
    IN  RvProxyPolicyMgrHandle       hPolicyMgr,
    IN  RvProxyPolicyMgrAppHandle    hAppMgr,
    IN  RvSipMsgHandle               hRecvReq,
    IN  RvSipMethodType              methodType,
    OUT RvBool*                      pbOpenDialog);
```

PARAMETERS

hPolicyMgr

The *ProxyPolicyMgr* handle.

hAppMgr

The handle of the application *ProxyPolicyMgr*.

hRecvReq

The handle of the received request.

methodType

The request method type (such as INVITE and so on.)

pbHandleLocal

The returned value according to the decision of the application:

RV_TRUE if the application opens a Dialog to handle the request. RV_FALSE if the application does not open a Dialog and handles the request as a Proxy Server.

RETURN VALUES

None.

RvProxyPolicyMgrHandleLocalPublishEv()

DESCRIPTION

Consults the application on whether the local Presence Server will handle an incoming PUBLISH request locally, or whether the PUBLISH request should be forwarded and then handled by the Proxy Server. This event is called when a new PUBLISH request is received for one of the SIP Server local domains.

SYNTAX

```
typedef void (RVCALLCONV *
RvProxyPolicyMgrHandleLocalPublishEv)(
    IN  RvProxyPolicyMgrHandle       hPolicyMgr,
    IN  RvProxyPolicyMgrAppHandle    hAppMgr,
    IN  RvSipMsgHandle               hPublishRequest,
    OUT RvBool*                      pbHandleLocal);
```

PARAMETERS

hPolicyMgr

The handle of the *ProxyPolicyMgr* handle.

hAppMgr

The handle of the application *ProxyPolicyMgr*.

hPublishRequest

The handle of the received PUBLISH request.

pbHandleLocal

The returned value according to the decision of the application. If RV_TRUE, handles the request locally using the Presence Server. If RV_FALSE, handles the request using the Proxy Server as all other requests. (The request may be forwarded or rejected.)

RETURN VALUES

None.

REMARKS

This function is relevant only if you have acquired the RADVISION SIP Events Server Add-on Module.

RvProxyPolicyMgrOtherURLRcvdEv()

DESCRIPTION

Consults the application about an incoming address whose URL type is not supported by the proxy. This callback is called whenever the SIP Server receives REGISTER, PUBLISH or SUBSCRIBE requests with unsupported URL in the Req-Uri.

The application should use this callback as follows:

1. Determine whether the Req-Uri indicates one of the proxy domains. If TRUE, the proxy will handle the request locally.
2. Rewrite *hAddress* in a canonical form that will be used in hash keys and further comparisons.

Note Rewriting *hAddress* is allowed only when the address was identified as the local domain of the proxy.

SYNTAX

```
typedef void (RVCALLCONV * RvProxyPolicyMgrOtherURLRcvdEv)(
    IN    RvProxyPolicyMgrHandle       hPolicyMgr,
    IN    RvProxyPolicyMgrAppHandle    hAppMgr,
    IN    RvSipMsgHandle               hMsg,
    INOUT RvSipAddressHandle           hAddress,
    OUT   RvBool                       *pbIsLocalDomain);
```

PARAMETERS

hPolicyMgr

The handle of the *ProxyPolicyMgr*.

hAppMgr

The handle of the application *ProxyPolicyMgr*.

hMsg

The handle of the request message that the server wishes to determine if it is local.

hAddress

The handle of the Req-Uri with unsupported URL.

pbIsLocalDomain

The returned value according to the decision of the application. RV_TRUE if this address indicates one of the domains of the proxy. Otherwise, RV_FALSE.

Return Values

None.

Proxy Core Object Events

The Proxy Core Object events are:

- RvProxyCoreObjectCreatedEv()
- RvProxyCoreObjStateChangeEv()
- RvProxyCoreObjMsgReceivedEv()
- RvProxyCoreObjMsgToSendEv()
- RvProxyCoreObjProcessResponseEv()
- RvProxyCoreObjInitiateResponseEv()
- RvProxyCoreObjSupplyRecordRouteForReqEv()
- RvProxyCoreObjRewriteRecordRouteInResp()
- RvProxyCoreObjInvalidResponseEv()
- RvProxyCoreObjRequestFinalDestResolvedEv()
- RvProxyCoreObjSLRequestFinalDestResolvedEv() **Deprecated**
- RvProxyCoreObjOtherURLRcvdEv()

RvProxyCoreObjectCreatedEv()

DESCRIPTION

Notifies the application that a new *ProxyCoreObj* has been created. The newly created *ProxyCoreObj* always assumes the IDLE state. The application can supply its handle to an associated application object. This handle will be supplied in any of the callbacks relevant to this object.

Note It is not permitted to call the Terminate() API function from this callback.

SYNTAX

```
typedef RvStatus (* RvProxyCoreObjectCreatedEv)(
    IN  RvProxyCoreObjHandle       hCoreObj,
    OUT RvProxyCoreObjAppHandle    *pAppCoreObj);
```

PARAMETERS

hCoreObj

The new *ProxyCoreObj* handle.

pAppCoreObj

The returned application handle for the *ProxyCoreObj*.

RETURN VALUES

Returns RvStatus. (Reserved for future use.)

RvProxyCoreObjStateChangeEv()

DESCRIPTION

Notifies the application of *ProxyCoreObj* state changes and the reason that caused changing the state. This event is probably the most useful of the events that the *ProxyCoreObj* reports.

SYNTAX

```
typedef void (* RvProxyCoreObjStateChangeEv)(
    IN RvProxyCoreObjHandle       hCoreObj,
    IN RvProxyCoreObjAppHandle    pAppCoreObj,
    IN RvProxyCoreObjState        eState,
    IN RvProxyCoreObjReason       eReason);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* handle.

hTranscOwner

The application handle for this *ProxyCoreObj*.

eState

The new state of the *ProxyCoreObj*.

eReason

The reason for the state change.

RETURN VALUES

None.

RvProxyCoreObjMsgReceivedEv()

DESCRIPTION

An event indicating that a *ProxyCoreObj*-related incoming message has been received.

Note It is not permitted to call the Terminate() API function from this callback.

SYNTAX

```
typedef RvStatus (* RvProxyCoreObjMsgReceivedEv)(
    IN RvProxyCoreObjHandle       hCoreObj,
    IN RvProxyCoreObjAppHandle    pAppCoreObj,
    IN RvProxyTranscHandle        hProxyTransc,
    IN RvProxyTranscAppHandle     pAppProxyTransc,
    IN RvSipMsgHandle             hRcvdMsg);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* handle.

pAppCoreObj

The application handle for this *ProxyCoreObj*.

hProxyTransc

The *ProxyCoreTransc* which received the message. This parameter is NULL on requests.

pAppProxyTransc

The application handle for the *ProxyCoreTransc*.

hRcvdMsg

The handle to the received message.

RETURN VALUES

Returns RvStatus. (The returned status is ignored in the current version.)

RvProxyCoreObjMsgToSendEv()

DESCRIPTION

Notifies the application that a *ProxyCoreObj*-related outgoing message is about to be sent.

Note It is not permitted to call the Terminate() API function from this callback.

SYNTAX

```
typedef RvStatus (* RvProxyCoreObjMsgToSendEv)(
    IN RvProxyCoreObjHandle       hCoreObj,
    IN RvProxyCoreObjAppHandle    pAppCoreObj,
    IN RvProxyTranscHandle        hProxyTransc,
    IN RvProxyTranscAppHandle     pAppProxyTransc,
    IN RvSipMsgHandle             hMsgToSend);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* handle.

hAppCoreObj

The application handle for this *ProxyCoreObj*.

hProxyTransc

The *ProxyCoreTransc* which is about to send the message. This parameter is NULL on requests.

pAppProxyTransc

The application handle for this *ProxyCoreTransc*.

hMsgToSend

The handle to the outgoing message.

RETURN VALUES

RV_Success if O.K. (The returned value is ignored in the current version.)

RvProxyCoreObjProcessResponseEv()

DESCRIPTION

Called when a response is received on one of the *ProxyCoreTransc* objects. This callback notifies the application of the action recommended by the SIP Server for the response received, such as forward the response or store the response in the response context. The application can choose another action, and if it is not contradicting the RFC or the internal logic, this is the action that will be taken for this response. Otherwise, the proxy continues. The decision as to what should be done with the response received depends on the response type and the *ProxyCoreObj* state.

SYNTAX

```
typedef void (* RvProxyCoreObjProcessResponseEv)(
    IN  RvProxyCoreObjHandle       hCoreObj,
    IN  RvProxyCoreObjAppHandle    hAppCoreObj,
    IN  RvProxyTranscHandle        hProxyTransc,
    IN  RvProxyTranscAppHandle     hAppProxyTransc,
    IN  RvSipMsgHandle             hMsgRcvd,
    IN  RvProxyCoreRespAction      recAction,
    OUT RvProxyCoreRespAction*     requiredAction);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* “owner” of the *ProxyCoreTransc*.

pAppCoreObj

The application handle for the *ProxyCoreObj*.

hProxyTransc

The *ProxyCoreTransc* handle. This parameter is NULL in stateless mode.

hAppProxyTransc

The application handle to the *ProxyCoreTransc* on which the response was received. This parameter is NULL in stateless mode, or if the application did not attach an object to the proxy transaction at RvProxyCoreTranscCreatedEv().

hMsgRcvd

The handle to the response message received.

recAction

The action the SIP Server recommends for this response, such as forward or discard.

requiredAction

The application can override the SIP Server recommended action and ask the SIP Server to perform another action on the response. The action that the application requires should be consistent with the class of the response. Otherwise, the proxy will ignore the application request and continue with the recommended action.

Return Values

None.

RvProxyCoreObjInitiateResponseEv()

DESCRIPTION

Called when a response is initiated by the SIP Server. The application can prevent the proxy from sending the response with the RVPROXY_RESPONSE_DISCARD action in the requiredAction argument. An example of a response that the proxy initiated is 408, when all transactions are terminated and no final response was yet received.

SYNTAX

```
typedef void (* RvProxyCoreObjInitiateResponseEv)(
    IN  RvProxyCoreObjHandle       hCoreObj,
    IN  RvProxyCoreObjAppHandle    hAppCoreObj,
    IN  RvUint16                   responseCode,
    IN  RvProxyCoreObjReason       eReason,
    IN  RvProxyCoreRespAction      recAction,
    OUT RvProxyCoreRespAction*     requiredAction);
```

PARAMETERS

hCoreObj

The handle to the *ProxyCoreObj*.

pAppCoreObj

The application handle for this *ProxyCoreObj*.

recAction

The action the proxy recommends for this response, such as forward and discard.

responseCode

The response code that the SIP Server initiated.

eReason

The reason for the response.

requiredAction

The application can override the SIP Server recommended action and ask the proxy to perform another action on the response. The action that the application requires should be consistent with the class of the response. Otherwise, the proxy will ignore the application request and continue with the recommended action.

Return Values

None.

RvProxyCoreObjSupplyRecordRouteForReqEv()

DESCRIPTION

Allows the application to supply Record Route header parameters. The Record Route header parameters supplied will be used for the outgoing request. Therefore, these parameters must be one of the domains of the proxy. If no parameters are set to the Record Route header, parameters from the configuration will be used to set the Record Route address. Configuration values will also be used in case the application does not implement this event.

SYNTAX

```
typedef void (* RvProxyCoreObjSupplyRecordRouteForReqEv)(
    IN RvProxyCoreObjHandle         hCoreObj,
    IN RvProxyCoreObjAppHandle      hAppCoreObj,
    IN RvSipRouteHopHeaderHandle    hRecordRouteHeader);
```

PARAMETERS

hCoreObj

The handle of the *ProxyCoreObj*.

pAppCoreObj

The application handle for this *ProxyCoreObj*.

hRecordRouteHeader

A Record Route header in which the application can set Record Route parameters. These parameters will be used in the outgoing request. The *lr* parameter will be added to the header.

RETURN VALUES

None.

RvProxyCoreObjRewriteRecordRouteInResp()

DESCRIPTION

Allows the application to rewrite the Record Route header parameters in a response. Configuration values will be used in case the application does not implement this event.

SYNTAX

```
typedef void (* RvProxyCoreObjRewriteRecordRouteInResp)(
    IN    RvProxyCoreObjHandle         hCoreObj,
    IN    RvProxyCoreObjAppHandle      hAppCoreObj,
    INOUT RvSipRouteHopHeaderHandle    hRecordRouteHeader);
```

PARAMETERS

hCoreObj

The handle of the *ProxyCoreObj*.

pAppCoreObj

The application handle for this *ProxyCoreObj*.

hRecordRouteHeader

The Record Route header which should be rewritten. This is a header that was identified by a unique ID as a header that was added by the SIP Server.

RETURN VALUES

None.

RvProxyCoreObjInvalidResponseEv()

DESCRIPTION

Called when an invalid response was received. The message will be discarded and the proxy will continue according to the *ProxyCoreObj* type (stateless or transaction stateful) and state (whether it is necessary to wait for additional responses).

SYNTAX

```
typedef void (* RvProxyCoreObjInvalidResponseEv)(
    IN RvProxyCoreObjHandle       hCoreObj,
    IN RvProxyCoreObjAppHandle    hAppCoreObj,
    IN RvProxyTranscHandle        hProxyTransc,
    IN RvProxyTranscAppHandle     hAppProxyTransc,
    IN RvSipMsgHandle             hMsgRcvd,
    IN RvProxyCoreTranscReason    eReason);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj*.

pAppCoreObj

The application handle for this *ProxyCoreObj*.

hProxyTransc

The *ProxyCoreTransc* that received the response.

hAppProxyTransc

The application handle to the *ProxyCoreTransc*.

hMsgRcvd

The handle of the invalid response message.

eReason

The reason for the invalid response.

Return Values

None.

RvProxyCoreObjRequestFinalDestResolvedEv()

DESCRIPTION

Notifies the application that the SIP Server "knows" the explicit IP address port and transport that is going to send the specified stateful request message. Before the callback is called, the SIP Server sets the local interface it will use to send the request. Choosing the address will be based on the first match according to address type, transport and port. The application can call RvProxyCoreTranscGetResolvedFinalDest() on this event to retrieve this information. Using this information, the application can base its decision on which local interface will be used to forward this request.

SYNTAX

```
typedef RvStatus (RVCALLCONV *
RvProxyCoreObjRequestFinalDestResolvedEv)(
    IN RvProxyCoreObjHandle       hCoreObj,
    IN RvProxyCoreObjAppHandle    hAppCoreObj,
    IN RvProxyTranscHandle        hProxyTransc,
    IN RvProxyTranscAppHandle     hAppProxyTransc,
    IN RvSipMsgHandle             hMsgToSend);
```

PARAMETERS

hCoreObj

The handle of the *ProxyCoreObj*.

pAppCoreObj

The application handle for this *ProxyCoreObj*.

hProxyTransc

The *ProxyCoreTransc* whose final destination was resolved.

pAppProxyTransc

The application handle for *hProxyTransc*.

hMsgToSend

Handle to the message that about to be send.

Return Values

Returns RvStatus.

RvProxyCoreObjSLRequestFinalDestResolvedEv()

This function is deprecated.

DESCRIPTION

Notifies the application that the SIP Server "knows" the explicit IP address port and transport to which it is going to send the specified stateless request message. Before this callback is called, the SIP Server sets the local interface it will use to send the request. Choosing the address will be based on the first match according to address type, transport and port. The application retrieves the final destination of the request (address transport and port) on this event. Using this information, the application can base it decision on which local interface will be used to forward this request. The application must return these values (address, port and transport) in the OUT parameters.

SYNTAX

```
typedef RvStatus (RVCALLCONV *
RvProxyCoreObjSLRequestFinalDestResolvedEv)(
    IN    RvProxyPolicyMgrHandle       hPolicyMgr,
    IN    RvProxyPolicyMgrAppHandle    hAppPolicyMgr,
    IN    RvSipMsgHandle               hMsgToSend,
    IN    RvSipTransport               eTransportType,
    IN    RvSipTransportAddressType    eAddressType,
    IN    const RvChar*                szDestAddress,
    IN    RvUint16                     destPort,
    INOUT RvChar*                      szLocalAddress,
    INOUT RvUint16*                    pLocalPort);
```

PARAMETERS

hPolicyMgr

The handle to the *ProxyPolicyMgr*.

hAppPolicyMgr

The application handle for the *ProxyPolicyMgr*.

hMsgToSend

The handle to the message that is about to be sent.

eTransportType

The transport type of the local and remote addresses.

eAddressType

The address type of the local and remote addresses.

szDestAddress

The destination IP address.

destPort

The destination port.

szLocalAddress

The retuned local IP address from which the message will be sent.

pLocalPort

The returned local port from which the message will be sent.

RETURN VALUES

Returns RvStatus.

RvProxyCoreObjOtherURLRcvdEv()

DESCRIPTION

Consults the application about an incoming address whose URL type is not supported by the proxy. This callback is called whenever a *ProxyCoreObj* receives a request with an unsupported URL in the Req-Uri. The application should use this callback as follows:

1. Determine whether or not the Req-Uri indicates one of the proxy domains. If TRUE, the proxy will search the Location Database to determine the request target.
2. Rewrite *hAddress* in a canonical form that will be used in hash keys and further comparisons.

Note Rewriting *hAddress* is allowed only when the address was identified as the local domain of the proxy.

SYNTAX

```
typedef void (RVCALLCONV * RvProxyCoreObjOtherURLRcvdEv)(
    IN    RvProxyCoreObjHandle       hCoreObj,
    IN    RvProxyCoreObjAppHandle    hAppCoreObj,
    IN    RvSipMsgHandle             hMsg,
    INOUT RvSipAddressHandle         hAddress,
    OUT   RvBool                     *pbIsLocalDomain);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj*.

pAppCoreObj

The application handle for this *ProxyCoreObj*.

hMsg

The handle of the request message that the proxy wishes to determine if it is local.

hAddress

The handle of the Req-Uri with unsupported URL.

pbIsLocalDomain

The returned value according to the decision of the application. RV_TRUE if this address indicates one of the domains of the proxy. Otherwise, RV_FALSE.

Return Values

None.

Stateful Proxy Core Object Events

The Stateful Proxy Core Object events are:

- RvProxyCoreTranscCreatedEv()
- RvProxyCoreTranscStateChangeEv()
- RvProxyCoreObjectInternalCreatedEv()
- RvProxyCoreObjBestRespToForwardEv()
- RvProxyCoreTranscOtherURLToSendEv()
- RvProxyCoreTranscNewConnInUseEv()

RvProxyCoreTranscCreatedEv()

DESCRIPTION

Notifies the application that a new *ProxyCoreTransc* object has been created. The application can supply its handle to the handle associated with the *ProxyCoreTransc* object. This handle will be supplied in any of the callbacks relevant to this object.

Note It is not permitted to call the Terminate() API function from this callback.

SYNTAX

```
typedef RvStatus (* RvProxyCoreTranscCreatedEv)(
    IN  RvProxyCoreObjHandle       hCoreObj,
    IN  RvProxyCoreObjAppHandle    hAppCoreObj,
    IN  RvProxyTranscHandle        hTransc,
    OUT RvProxyTranscAppHandle     *pAppTranscObj);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* handle that “owns” the newly created *ProxyCoreTransc*.

pAppCoreObj

The application handle attached to the *ProxyCoreObj*.

hTransc

The newly created *ProxyCoreTransc* handle.

pAppTranscObj

The returned application handle for the created *ProxyCoreTransc*.

RETURN VALUES

Returns RvStatus. (Reserved for future use.)

RvProxyCoreTranscStateChangeEv()

DESCRIPTION

Notifies the application that a *ProxyCoreTransc* object has changed state.

SYNTAX

```
typedef void (* RvProxyCoreTranscStateChangeEv)(
    IN RvProxyCoreObjHandle       hCoreObj,
    IN RvProxyCoreObjAppHandle    pAppCoreObj,
    IN RvProxyTranscHandle        hTransc,
    IN RvProxyTranscAppHandle     pAppTranscObj,
    IN RvProxyCoreTranscState     eState,
    IN RvProxyCoreTranscReason    eReason);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* handle that contains the *ProxyCoreTransc*.

pAppCoreObj

The application handle attached to the *ProxyCoreObj*.

hTransc

The *ProxyCoreTransc* handle whose state is about to be changed.

pAppTranscObj

The application handle for the *ProxyCoreTransc*.

eState

The new state of the *ProxyCoreTransc.*

eReason

The reason for the state change.

RETURN VALUES

None.

RvProxyCoreObjectInternalCreatedEv()

DESCRIPTION

Notifies the application that the SIP Server created a new *ProxyCoreObj*, while cancelling another outgoing request. The newly created *ProxyCoreObj* assumes the IDLE state.

Note It is not permitted to call the Terminate() API function from this callback.

SYNTAX

```
typedef RvStatus (* RvProxyCoreObjectInternalCreatedEv)(
    IN  RvProxyCoreObjHandle       hCoreObj,
    OUT RvProxyCoreObjAppHandle    *pAppCoreObj
    IN  RvProxyTranscHandle        hProxyTransc
    OUT RvProxyTranscAppHandle     *pAppProxyTransc);
```

PARAMETERS

hCoreObj

The created *ProxyCoreObj* handle.

pAppCoreObj

The returned application handle for the *ProxyCoreObj* object.

hProxyTransc

The created *ProxyCoreTransc*.

pAppProxyTransc

The returned application handle for the *ProxyCoreTransc*.

RETURN VALUES

Returns RvStatus. (Reserved for future use.)

RvProxyCoreObjBestRespToForwardEv()

DESCRIPTION

If no response was forwarded, and all *ProxyCoreTransc* objects have been terminated, the application should choose one of the responses received, called "best response" and forward it as final response.

This callback gives the application notification about the response that was chosen by the proxy as its best response. The application can replace the message and set another message handle to send as best response. If the application set another message as best response, the SIP Server copies this message and uses it as the best response to forward.

Note This event is called after processing the best response chosen (aggregating the relevant headers). If the application chooses to replace the message, the message is sent as set by the application without any changes (as is).

SYNTAX

```
typedef void (* RvProxyCoreObjBestRespToForwardEv)(
    IN    RvProxyCoreObjHandle       hProxyCoreObj,
    IN    RvProxyCoreObjAppHandle    hAppCoreObj,
    IN    RvProxyListHandle          hRespContextList,
    INOUT RvSipMsgHandle*            phBestRespMsg);
```

PARAMETERS

hProxyCoreObj

The *ProxyCoreObj* whose client transactions were terminated.

hAppCoreObj

The application handle of *ProxyCoreObj*.

hRespContextList

The handle to the list of responses received for this response context. The application can choose another message from the context as best response.

phBestRespMsg

The pointer of the handle of the best response message.

RETURN VALUES

None.

RvProxyCoreTranscOtherURLToSendEv()

DESCRIPTION

Notifies the application that a message needs to be sent (in stateful mode), and the destination address is a URL type that the proxy does not currently support. In order for the message to be sent, the application must convert this URL into a SIP URL (supply the result in *hSipURLAddress*).

SYNTAX

```
typedef RvStatus (RVCALLCONV *
RvProxyCoreTranscOtherURLToSendEv)(
    IN  RvProxyCoreObjHandle       hCoreObj,
    IN  RvProxyCoreObjAppHandle    hAppCoreObj,
    IN  RvProxyTranscHandle        hProxyTransc,
    IN  RvProxyTranscAppHandle     hAppProxyTransc,
    IN  RvSipMsgHandle             hMsg,
    IN  RvSipAddressHandle         hAddress,
    OUT RvSipAddressHandle         hSipURLAddress,
    OUT RvBool                     *bAddressResolved);
```

PARAMETERS

hCoreObj

The handle of the *ProxyCoreObj*.

pAppCoreObj

The application handle for this *ProxyCoreObj*.

hProxyTransc

The *ProxyCoreTransc* that is about to send a message.

pAppProxyTransc

The application handle for this *ProxyCoreTransc*.

hMsg

The message to send, which includes the other URL address.

hAddress

The handle to the unsupported address to be converted.

hSipURLAddress

The handle to the SIP URL address that the application should supply—this is an empty address object that the application should fill.

bAddressResolved

RV_TRUE if the SIP URL address was filled. Otherwise, RV_FALSE.

Return Values

Returns RvStatus.

If an error is returned, the message is not sent.

RvProxyCoreTranscNewConnInUseEv()

DESCRIPTION

Indicates that the *ProxyCoreTransc* is now using a new connection. The connection can be completely new or a suitable connection that was found in the hash.

SYNTAX

```
typedef RvStatus (RVCALLCONV
*RvProxyCoreTranscNewConnInUseEv) (
    IN RvProxyCoreObjHandle              hCoreObj,
    IN RvProxyTranscHandle               hProxyTransc,
    IN RvProxyTranscAppHandle            hAppProxyTransc,
    IN RvSipTransportConnectionHandle    hConn,
    IN RvBool                            bNewConnCreated);
```

PARAMETERS

hCoreOb

The handle of the *ProxyCoreObj*.

pAppCoreObj

The application handle for this *ProxyCoreObj*.

hProxyTransc

The *ProxyCoreTransc* that is about to use a new connection.

pAppProxyTransc

The application handle for this *ProxyCoreTransc*.

hConn

The connection handle.

bNewConnCreated

RV_TRUE if the connection is a newly created connection.

RV_FALSE if the connection was found in the connection hash.

Return Values

Returns RvStatus.

Stateless Proxy Core Object Events

The Stateless Proxy Core Object events are:

- RvProxyCoreTrxCreatedEv()
- RvProxyCoreTrxStateChangeEv()
- RvProxyCoreTrxOtherURLToSendEv()

RvProxyCoreTrxCreatedEv()

DESCRIPTION

Notifies the application that a new *ProxyTrx* was created. The application can supply its handle to an associated application object. This handle will be supplied in any of the callbacks relevant for this object.

Note It is not permitted to call the Terminate() API function from this callback.

SYNTAX

```
typedef RvStatus (RVCALLCONV * RvProxyCoreTrxCreatedEv)(
    IN  RvProxyCoreObjHandle       hCoreObj,
    IN  RvProxyCoreObjAppHandle    hAppCoreObj,
    IN  RvProxyTrxHandle           hProxyTrx,
    OUT RvProxyTrxAppHandle        *pAppTrxObj);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* handle which contains the newly created *ProxyTrx* object.

pAppCoreObj

The application handle attached to the *ProxyCoreObj*.

hProxyTrx

The newly created *ProxyTrx* handle.

pAppTrxObj

The returned application handle for the created transmitter.

RETURN VALUES

Reserved for future use.

RvProxyCoreTrxStateChangeEv()

DESCRIPTION

Notifies the application that a *ProxyCoreObj* changed its state.

SYNTAX

```
typedef void (RVCALLCONV * RvProxyCoreTrxStateChangeEv)(
    IN RvProxyCoreObjHandle       hCoreObj,
    IN RvProxyCoreObjAppHandle    pAppCoreObj,
    IN RvProxyTrxHandle           hProxyTrx,
    IN RvProxyTrxAppHandle        pAppTrxObj,
    IN RvProxyCoreTrxState        eState,
    IN RvProxyCoreTrxReason       eReason,
    IN RvSipMsgHandle             hMsg,
    IN void*                      pExtraInfo);
```

PARAMETERS

hCoreObj

The *ProxyCoreObj* handle which contains the *ProxyTrx* that is about to be terminated.

pAppCoreObj

The application handle attached to the *ProxyCoreObj*.

hProxyTrx

The *ProxyTrx* handle.

pAppTrxObj

The application handle for the *ProxyTrx*.

eState

The new state of the *ProxyTrx*.

pExtraInfo

Specific information for future states.

hMsg

When the state relates to the outgoing message, the message is supplied.

RETURN VALUES

None.

RvProxyCoreTrxOtherURLToSendEv()

DESCRIPTION

Notifies the application that a message needs to be sent (in stateless mode), and the destination address is a URL type that the proxy does not currently support. In order for the message to be sent, the application must convert this URL into a SIP URL (supply the result in *hSipURLAddress*).

SYNTAX

```
typedef RvStatus (RVCALLCONV *
RvProxyCoreTrxOtherURLToSendEv)(
    IN  RvProxyCoreObjHandle       hCoreObj,
    IN  RvProxyCoreObjAppHandle    hAppCoreObj,
    IN  RvProxyTrxHandle           hProxyTrx,
    IN  RvProxyTrxAppHandle        hAppProxyTrx,
    IN  RvSipMsgHandle             hMsg,
    IN  RvSipAddressHandle         hAddress,
    OUT RvSipAddressHandle         hSipURLAddress,
    OUT RvBool                     *bAddressResolved);
```

PARAMETERS

hCoreObj

The handle of the *ProxyCoreObj*.

pAppCoreObj

The application handle for this *ProxyCoreObj*.

hProxyTrx

The *ProxyCoreTrx* that is about to send a message.

pAppProxyTrx

The application handle for this *ProxyCoreTrx*.

hMsg

The message to send, which includes the other URL address.

hAddress

The handle to the unsupported address to be converted.

hSipURLAddress

The handle to the SIP URL address that the application should supply—this is an empty address object that the application should fill.

bAddressResolved

RV_TRUE if the SIP URL address was filled. Otherwise, RV_FALSE.

Return Values

Returns RvStatus.

If an error is returned, the message is not sent.

14

Register Server Type Definitions

What's in this Section

This section includes the Register Server type definitions defined in the *RvProxyRegServerTypes.h* header file.

The type definitions in this section are:

- Handles
- Structures and Enumerations
- Component Interface Type Definitions for LocationDB
- Component Interface Type Definitions for PASC Interface
- Callback Functions

HANDLES

The handles are:

- RvProxyRegServerMgrHandle
- RvProxyRegServerMgrAppHandle
- RvProxyRegServerObjHandle
- RvProxyRegServerObjAppHandle
- RvProxyRegServerRecordHandle
- RvProxyRegServerBindingHandle
- RvProxyRegServerKeyHandle

RvProxyRegServerMgrHandle

DESCRIPTION

The declaration of a Proxy Register Server Manager (*RegServerMgr*) object handle. This handle is needed for Register Server Manager API functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyRegServerMgrHandle);
```

RvProxyRegServerMgrAppHandle

DESCRIPTION

The declaration of an application *RegServerMgr* object handle. This handle can be attached to the SIP Server *RegServerMgr* object.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyRegServerMgrAppHandle);
```

RvProxyRegServerObjHandle

DESCRIPTION

The declaration of a Register Server object (*RegServerObj*) handle. This handle is needed for Register Server Object API functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyRegServerObjHandle);
```

RvProxyRegServerObjAppHandle

DESCRIPTION

The declaration of an application handle to a *RegServerObj*. The application uses this handle in order to associate a *RegServerObj* with an application *RegServerObj*. The application gives this handle when a new *RegServerObj* is created. The SIP Server will give this handle back to the application in each callback function.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyRegServerObjAppHandle);
```

RvProxyRegServerRecordHandle

DESCRIPTION

The declaration of a Register Server Record object (*RegServerRecord*) handle. This handle is needed for Register Server Object API functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyRegServerRecordHandle);
```

RvProxyRegServerBindingHandle

DESCRIPTION

The declaration of a Register Server Binding object (*RegServerBinding*) handle. This handle is needed for Register Server Object API functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyRegServerBindingHandle);
```

RvProxyRegServerKeyHandle

DESCRIPTION

The declaration of a Register Server Key object (*RegServerKey*) handle. This handle is needed for Register Server Object API functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyRegServerKeyHandle);
```

STRUCTURES AND ENUMERATIONS

The structures and enumerations are:

- RvProxyRegServerObjState
- RvProxyRegServerObjReason
- RvProxyRegServerEvHandler

RvProxyRegServerObjState

DESCRIPTION

The state of the *RegServerObj*.

SYNTAX

```
typedef enum{
    RVPROXY_REGSERVER_OBJ_STATE_UNDEFINED = -1,
    RVPROXY_REGSERVER_OBJ_STATE_IDLE,
    RVPROXY_REGSERVER_OBJ_STATE_REQ_RCVD,
    RVPROXY_REGSERVER_OBJ_STATE_AUTHENTICATING,
    RVPROXY_REGSERVER_OBJ_STATE_VALID_REG,
    RVPROXY_REGSERVER_OBJ_STATE_INVALID_REG,
    RVPROXY_REGSERVER_OBJ_STATE_FINAL_RESPONSE_SENT,
    RVPROXY_REGSERVER_OBJ_STATE_BEFORE_TERMINATE,
    RVPROXY_REGSERVER_OBJ_STATE_TERMINATE,
#ifdef RVSIPSERVER_LDAP
    RVPROXY_REGSERVER_OBJ_STATE_EXECUTING_QUERY
#endif /*RVSIPSERVER_LDAP*/
}RvProxyRegServerObjState;
```

PARAMETERS

RVPROXY_REGSERVER_OBJ_STATE_IDLE

The IDLE state is the initial state of the Register Server Object state machine. When a *RegServerObj* is created, the object assumes the IDLE state. From this state, registration validity checks are done on the Registration request.

RVPROXY_REGSERVER_OBJ_STATE_REQ_RCVD

This state is reached when the transaction changes its state to REQ_RCVD. In this state, the received request is validated and authenticated if needed.

RVPROXY_REGSERVER_OBJ_STATE_AUTHENTICATING

The *RegServerObj* reaches this state after validating the received message, if authentication is needed. If authentication succeeds, the *RegServerObj* changes its state to VALID_REG. If authentication fails, the *RegServerObj* changes its state to INVALID_REG.

RVPROXY_REGSERVER_OBJ_STATE_VALID_REG

This state occurs after the *RegServerObj* has checked the registration validity. From this state, the application should call the Registration Accept/Reject function.

RVPROXY_REGSERVER_OBJ_STATE_INVALID_REG

This state occurs when the Registration request was found to be invalid by the *RegServerObj*. From this state, the application should call the Registration Reject function.

RVPROXY_REGSERVER_OBJ_STATE_FINAL_RESPONSE_SENT

This state occurs after a final response was sent.

RVPROXY_REGSERVER_OBJ_STATE_BEFORE_TERMINATE

This is the last state before final termination of the *RegServerObj*. This is the last chance for the application to retrieve information from the *RegServerObj*.

RVPROXY_REGSERVER_OBJ_STATE_TERMINATE

After RVPROXY_REGSERVER_FINAL_RESPONSE_SENT, the *RegServerObj* waits for the related *ProxyCoreObj* to be terminated. Once this happens, the *RegServerObj* is terminated as well.

RVPROXY_REGSERVER_OBJ_STATE_EXECUTING_QUERY

When the *RegServerObj* is configured to work with a remote database, it instructs the remote database to perform lookup, add, update and remove queries. While instructing to perform those queries, the *RegServerObj* assumes the EXECUTING_QUERTY state. The *RegServerObj* remains in this state until the queries are executed.

RvProxyRegServerObjReason

DESCRIPTION

The reason for the changing state of the *RegServerObj*, or for rejecting/ accepting a Registration request.

SYNTAX

```
typedef enum{
 RVPROXY_REGSERVER_OBJ_REASON_UNDEFINED = -1,
 RVPROXY_REGSERVER_OBJ_REASON_ERROR,
 RVPROXY_REGSERVER_OBJ_REASON_REGISTRATION_VALID,
 RVPROXY_REGSERVER_OBJ_REASON_INVALID_TO_HEADER,
 RVPROXY_REGSERVER_OBJ_REASON_INVALID_STAR_CONTACT_HEADER,
 RVPROXY_REGSERVER_OBJ_REGISTRATION_SUCCEEDED,
 RVPROXY_REGSERVER_OBJ_REGISTRATION_FAILED,
 RVPROXY_REGSERVER_OBJ_REASON_CORE_OBJECT_ERROR,
 RVPROXY_REGSERVER_OBJ_REASON_NORMAL_TERMINATION,
 RVPROXY_REGSERVER_OBJ_REJECT_FINAL_RESPONSE_SENT,
 RVPROXY_REGSERVER_OBJ_UPDATE_UNFINISHED,
 RVPROXY_REGSERVER_OBJ_UPDATE_FAILED,
 RVPROXY_REGSERVER_OBJ_REASON_INVALID_EXPIRE,
 RVPROXY_REGSERVER_OBJ_REASON_TERMINATED_BY_APPLICATION,
 RVPROXY_REGSERVER_OBJ_REASON_AUTHENTICATION_SUCCEED,
 RVPROXY_REGSERVER_OBJ_REASON_AUTHENTICATION_FAILED,
 RVPROXY_REGSERVER_OBJ_REASON_AUTHENTICATION_ERROR,
 RVPROXY_REGSERVER_OBJ_REASON_FAILURE_DISREGARDED,
 RVPROXY_REGSERVER_OBJ_REASON_REQUEST_UNSUPPORTED_EXTENSION,
 RVPROXY_REGSERVER_OBJ_REASON_SIPS_INCONSISTENCY,
}RvProxyRegServerObjReason;
```

RvProxyRegServerEvHandler

DESCRIPTION

A structure with function pointers to module callbacks. This structure is used to set the application implementations using RvProxyRegServerMgrSetEvHandler().

SYNTAX

```
typedef struct{
    RvProxyRegServerObjCreatedEv
                        pfnEvRegServerObjCreated;
    RvProxyRegServerObjMsgRcvdEv
                        pfnEvRegServerObjReqMsgRcvd;
    RvProxyRegServerObjStateChangedEv
                        pfnEvRegServerObjStateChanged;
    RvProxyRegServerObjRespMsgToSendEv
                        pfnEvRegServerObjRespMsgToSend;
    RvProxyRegServerObjDBUpdateFailed
                        pfnEvRegServerObjDBUpdateFailed;
    RvProxyRegServerObjOtherURLRcvdEv
                        pfnEvRegServerObjOtherURLRcvd;
    RvProxyRegServerObjFinalDestResolvedEv
                        pfnEvRegServerObjFinalDestResolved;
}RvProxyRegServerEvHandler;
```

PARAMETERS

pfnEvRegServerObjCreated

Called when a new *RegServerObj* is created. The application can attach an owner that will be associated with the created *RegServerObj*.

pfnEvRegServerObjReqMsgRcvd

Notifies the application when a request message is received. The application can modify the request before it is processed.

pfnEvRegServerObjStateChanged

Notifies the application about a change in the *RegServerObj* state

pfnEvRegServerObjRespMsgToSend

Notifies the application when a response is about to be sent. The application can modify the response before it is sent.

pfnEvRegServerObjDBUpdateFailed

Notifies the application when updating the database has failed. The application should choose whether to ignore the failure and continue with accepting the registration, or reject the registration due to failure.

pfnEvRegServerObjOtherURLRcvd

Consults the application about an incoming address whose URL type is not supported by the SIP Server.

pfnEvRegServerObjFinalDestResolved

Notifies the application that the *RegServerObj* is about to send a message after the destination address is resolved.

Component Interface Type Definitions for LocationDB

The type definitions are:

- RvProxyLocationDBMgrHandle
- RvProxyLocationDBInterface
- RvProxyLocationDBUpdateRecord()
- RvProxyLocationDBInsertRecord()
- RvProxyLocationDBLookUpRecord()
- RvProxyLocationDBRemoveRecord()

RvProxyLocationDBMgrHandle

DESCRIPTION

The handle to the *LocationDB* that the application constructed.

SYNTAX

```
RV_DECLARE_HANDLE(RvProxyLocationDBMgrHandle);
```

RvProxyLocationDBInterface

DESCRIPTION

A structure with function pointers to the Location Database Interface functions that should be implemented by the Location Database Server Component. This structure is used to set the interface functions to the *SipServerMgr* using RvProxyRegServerMgrLocationDBSetInterface().

SYNTAX

```
typedef struct{
    RvProxyLocationDBUpdateRecord pfnLocationDBUpdate;
    RvProxyLocationDBInsertRecord pfnLocationDBInsert;
    RvProxyLocationDBLookupRecord pfnLocationDBLookup;
    RvProxyLocationDBRemoveRecord pfnLocationDBRemoveRecord;
}RvProxyLocationDBInterface;
```

PARAMETERS

pfnLocationDBUpdate

Updates a registration request.

pfnLocationDBInsert

Inserts a record into the *LocationDB*.

pfnLocationDBLookup

Checks the registration details of the user.

pfnLocationDBRemoveRecord

Removes a record from the *LocationDB*.

RvProxyLocationDBUpdateRecord()

DESCRIPTION

Updates a record in the *LocationDB*. This function expects an updated record. If a record for this key already exists, it is replaced with the updated record. If no record exists, the updated record is added as a new record. A *LocationDB* record will be created from the *RegServerObj* record argument.

SYNTAX

```
typedef RvStatus (* RvProxyLocationDBUpdateRecord) (
    IN    RvProxyLocationDBMgrHandlev     hLocationDBMgr,
    IN    RvProxyRegServerKeyHandle       hRecordKey,
    INOUT RvProxyRegServerRecordHandle    hUpdatedRecord,
    OUT   RvSipServerQueryStatus*         eStatus);
```

PARAMETERS

hLocationDBMgr

The handle to the *LocationDBMgr*.

hRegServerKey

The handle to the key that the *RegServerObj* created.

hRegServerRecord

The handle to the registration record that the *RegServerObj* created. This is the updated record that should be inserted into the *LocationDB* for this key. A *LocationDB* record will be created from the *RegServerObj* record.

eStatus

For asynchronous mode, Finished or Pending.

RETURN VALUES

Returns RvStatus.

RvProxyLocationDBInsertRecord()

DESCRIPTION

Expects a record with new bindings that should be added to the binding list in the existing record. If a record exists for this key, the bindings are added to its list as is. If no record exists for this key, the record is added to the Registration Table.

SYNTAX

```
typedef RvStatus (* RvProxyLocationDBInsertRecord) (
    IN    RvProxyLocationDBMgrHandle      hLocationDBMgr,
    IN    RvProxyRegServerKeyHandle       hRecordKey,
    INOUT RvProxyRegServerRecordHandle    hRecord,
    OUT   RvSipServerQueryStatus*         eStatus);
```

PARAMETERS

hLocationDBMgr

The handle to the *LocationDBMgr*.

hRegServerKey

The handle to key that the *RegServerObj* created.

hRecord

The record with the new bindings to be added to the *LocationDB*.

eStatus

For asynchronous mode, Finished or Pending. (In the current version, only the finished status is legal).

RETURN VALUES

Returns RvStatus.

RvProxyLocationDBLookUpRecord()

DESCRIPTION

Looks up an existing record by key. If a record is found, the binding list is checked for expired contacts. If expired contacts are found, they are removed from the list. A *RegServerObj* record is created from the *LocationDB* record holding the lookup results. After using the lookup results, the application should free the *RegServerObj* record that was created.

If no record is found, NULL is returned.

SYNTAX

```
typedef RvStatus (* RvProxyLocationDBLookupRecord)(
    IN    RvProxyLocationDBMgrHandle      hLocationDBMgr,
    IN    RvProxyRegServerMgrHandle       hRegServerMgr,
    IN    RvProxyRegServerKeyHandle       hRecordKey,
    INOUT RvProxyRegServerRecordHandle    hRecord,
    OUT   RvSipServerQueryStatus*         eStatus);
```

PARAMETERS

hLocationDBMgr

The handle to the *LocationDBMgr*.

hRegServerMgr

The handle to the *RegServerMgr*.

hRegServerKey

The handle to the *RegServerObj* key.

phRecord

The handle to the *RegServerObj* record created with data found in the *LocationDB*.

eStatus

For asynchronous mode, Finished or Pending. (In the current version, only the finished status is legal).

RETURN VALUES

Returns RvStatus.

RvProxyLocationDBRemoveRecord()

DESCRIPTION

Removes the registration record and de-allocates it. This function looks for the record in the Registration Table by the key. If found, removes it. All the bindings of the record are removed and de-allocated.

SYNTAX

```
typedef RvStatus (* RvProxyLocationDBRemoveRecord)(
    IN  RvProxyLocationDBMgrHandle    hLocationDBMgr,
    IN  RvProxyRegServerKeyHandle     hRecordKey,
    OUT RvSipServerQueryStatus*       eStatus);
```

PARAMETERS

hLocationDBMgr

The handle to the *LocationDBMgr*.

hRegServerKey

The handle to the *RegServerObj* key which is used for finding the record to be removed.

eStatus

For asynchronous mode, Finished or Pending. (In the current version, only the finished status is legal).

RETURN VALUES

Returns RvStatus.

Component Interface Type Definitions for PASC Interface

The type definitions are:

- RvProxyRegServerPASCInterface
- RvSipServerRegisterUpdate

RvProxyRegServerPASCInterface

DESCRIPTION

A structure with function pointers to the PASC Interface functions that the PASC Component should implement. This structure is used to set the interface functions to the *RegServerMgr* using RvProxyRegServerMgrPASCSetInterface().

SYNTAX

```
typedef struct{
    RvSipServerRegisterUpdate    pfnPASCRegisterUpdate;
}RvProxyRegServerPASCInterface;
```

PARAMETERS

pfnPASCRegisterUpdate

Updates the Presence status.

RvSipServerRegisterUpdate

DESCRIPTION

Notification from the *RegServerObj* about a REGISTER request that was accepted and updated by the *RegServerObj*. This REGISTER request is used for creating Presence information and notifying the subscribed Watchers.

SYNTAX

```
typedef RvStatus (*RvSipServerRegisterUpdate) (
    IN void*                            hPASCMgr,
    IN RvProxyRegServerRecordHandle     hRegRecord);
```

PARAMETERS

hPASCMgr

The handle to the *PASCMgr*.

hRegRecord

The registration record that the *RegServerObj* updated.

CALLBACK FUNCTIONS

The callback functions are:

- RvProxyRegServerObjCreatedEv()
- RvProxyRegServerObjStateChangedEv()
- RvProxyRegServerObjMsgRcvdEv()
- RvProxyRegServerObjRespMsgToSendEv()
- RvProxyRegServerObjDBUpdateFailedEv()
- RvProxyRegServerObjOtherURLRcvdEv()
- RvProxyRegServerObjFinalDestResolvedEv()

RvProxyRegServerObjCreatedEv()

DESCRIPTION

Called when a new *RegServerObj* is created. The application can attach an owner object that will be associated with the created *RegServerObj*.

Note It is not permitted to call the Terminate() API function from this callback.

SYNTAX

```
typedef RvStatus (* RvProxyRegServerObjCreatedEv)(
    IN  RvProxyRegServerMgrHandle        hRegServerMgr,
    IN  RvProxyRegServerMgrAppHandle     hRegServerAppMgr,
    IN  RvProxyRegServerObjHandle        hRegServerObj,
    OUT RvProxyRegServerObjAppHandle*    pAppRegServer);
```

PARAMETERS

hRegServerMgr

The handle to the *RegServerMgr*.

hRegServerAppMgr

The handle to the application owner of the *RegServerMgr*.

hRegServerObj

The created *RegServerObj*.

pAppRegServer

The pointer to the new application owner of the *RegServerObj*.

RETURN VALUES

Returns RvStatus. (Reserved for future use.)

RvProxyRegServerObjStateChangedEv()

DESCRIPTION

Notifies the application about a change in the *RegServerObj* state.

SYNTAX

```
typedef void (* RvProxyRegServerObjStateChangedEv)(
    IN RvProxyRegServerObjHandle    hRegServerObj,
    IN RvProxyRegServerAppHandle    hAppRegServer,
    IN RvProxyRegServerObjState     eState,
    IN RvProxyRegServerObjReason    eReason);
```

PARAMETERS

hRegServerObj

The *RegServerObj* whose state is changed.

hAppRegServer

The application owner of the *RegServerObj*.

eState

The new state.

eReason

The reason for changing the state.

RETURN VALUES

None.

RvProxyRegServerObjMsgRcvdEv()

DESCRIPTION

Notifies the application when a Request message is received. The application can modify the request before it is being processed.

Note It is not permitted to call the Terminate() API function from this callback.

SYNTAX

```
typedef RvStatus (* RvProxyRegServerObjMsgRcvdEv)(
    IN RvProxyRegServerObjHandle       hRegServerObj,
    IN RvProxyRegServerObjAppHandle    hAppRegServer,
    IN RvSipMsgHandle                  hMsgRcvd);
```

PARAMETERS

hRegServerObj

The *RegServerObj* whose state is changed.

hAppRegServer

The application owner of the *RegServerObj*.

hMsgToSend

The response message to be sent.

RETURN VALUES

Returns RvStatus.

RvProxyRegServerObjRespMsgToSendEv()

DESCRIPTION

Notifies the application when a response is about to be sent. The application can access the response before it is being sent. The application must not terminate the *RegServerObj* while handling this event.

Note It is not permitted to call the Terminate() API function from this callback.

SYNTAX

```
typedef RvStatus (* RvProxyRegServerObjRespMsgToSendEv)(
    IN RvProxyRegServerObjHandle    hRegServerObj,
    IN RvProxyRegServerAppHandle    hAppRegServer,
    IN RvSipMsgHandle               hMsgToSend);
```

PARAMETERS

hRegServerObj

The *RegServerObj* handling this registration.

hAppRegServer

The application owner of the *RegServerObj*.

hMsgToSend

The response message that is about to be sent.

RETURN VALUES

If OK, returns RV_Success. (The returned value is ignored in the current version.)

RvProxyRegServerObjDBUpdateFailedEv()

DESCRIPTION

Notifies the application when updating the Location Database has failed. The application should choose whether to ignore the failure and continue with accepting the registration and sending a 200 response (bAcceptReg = TRUE) or reject the registration due to failure (bAcceptReg = FALSE). If the application chooses to reject the registration, a 500 response is sent to the client.

SYNTAX

```
typedef void (* RvProxyRegServerObjDBUpdateFailedEv)(
    IN  RvProxyRegServerObjHandle    hRegServerObj,
    IN  RvProxyRegServerAppHandle    hAppRegServer,
    IN  RvProxyRegServerObjReason    eReason,
    OUT RvBool*                      bAcceptReg);
```

PARAMETERS

hRegServerObj

The *RegServerObj* handling the registration.

hAppRegServer

The application owner of the *RegServerObj*.

eReason

The reason for the update failure.

bAcceptReg

If TRUE, sends a 200 response and ignores the failure. Otherwise, sends a 500 response.

RETURN VALUES

None.

RvProxyRegServerObjOtherURLRcvdEv()

DESCRIPTION

Consults the application about an incoming address whose URL type is not supported by the SIP Server. This callback is called whenever the Register Server receives a REGISTER request with an unsupported URL in the To header.

The application should use this callback as follows:

1. Determine whether or not the To header indicates one of the proxy domains. If TRUE, the Register Server will perform registration.
2. Rewrite *hAddress* in a canonical form that will be used in hash keys and further comparisons.

Note Rewriting *hAddress* is allowed only when the address was identified as local domain of the proxy.

SYNTAX

```
typedef void (RVCALLCONV *
RvProxyRegServerObjOtherURLRcvdEv)(
    IN    RvProxyRegServerObjHandle       hRegServerObj,
    IN    RvProxyRegServerObjAppHandle    hAppRegServer,
    IN    RvSipMsgHandle                  hMsg,
    INOUT RvSipAddressHandle              hAddress,
    OUT   RvBool                          *pbIsLocalDomain);
```

PARAMETERS

hRegServerObj

The *RegServerObj* handling the registration.

hAppRegServer

The application owner of the *RegServerObj*.

hMsg

The handle of the request message that the Register Server wishes to determine if it is local.

hAddress

The handle of the To address with unsupported URL.

pbIsLocalDomain

The returned value according to the decision of the application. RV_TRUE if this address indicates one of the domains of the proxy. Otherwise, RV_FALSE.

Return Values

None.

RvProxyRegServerObjFinalDestResolvedEv()

DESCRIPTION

Notifies the application that the *RegServerObj* is about to send a message after the destination address was resolved. This callback supplies the final message object and the transaction that is responsible for sending this message. Changes in the message at this stage will not effect the destination address.

SYNTAX

```
typedef RvStatus (* RvProxyRegServerObjFinalDestResolvedEv)(
    IN    RvProxyRegServerObjHandle       hRegServerObj,
    IN    RvProxyRegServerObjAppHandle    hAppRegServer,
    IN    RvSipMsgHandle                  hMsg);
```

PARAMETERS

hRegServerObj

The *RegServerObj* handling this transaction.

hMsgToSend

The message about to be sent by the transaction.

RETURN VALUES

Returns RvStatus.

15

SERVER AUTHENTICATION TYPE DEFINITIONS

WHAT'S IN THIS SECTION

This section includes the following Server Authentication type definitions as defined in the *RvSipServerAuthTypes.h* header file.

The type definitions in this section are:

- Handles
- Structures and Enumerations
- Component Interface Type Definitions for Security Server Components

HANDLES

The handles are:

- RvSipServerAuthMgrHandle()
- RvSipServerAuthMgrAppHandle()
- RvSipServerAuthObjHandle()
- RvSipServerAuthObjAppHandle()

RvSipServerAuthMgrHandle()

DESCRIPTION

The declaration of a Server Authentication Manager object (*ServerAuthMgr*) handle. This handle is needed for Server Authentication API functions. This handle can be accessed with RvSipServerMgrGetAuthMgrHandle().

SYNTAX

```
RV_DECLARE_HANDLE(RvSipServerAuthMgrHandle);
```

RvSipServerAuthMgrAppHandle()

DESCRIPTION

The declaration of an application-associated handle to the Server Authentication Manager. The application uses this handle to associate the *ServerAuthMgr* with the application-associated object. The application supplies this handle in the different events from the Server Authentication Manager.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipServerAuthMgrAppHandle);
```

RvSipServerAuthObjHandle()

DESCRIPTION

The declaration of handle to a Server Authentication object (*ServerAuth*). The *ServerAuth* is used to authenticate a received request, using any algorithm known to the component. This handle should be supplied when calling the Server Authentication API functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipServerAuthObjHandle);
```

RvSipServerAuthObjAppHandle()

DESCRIPTION

The declaration of an application handle to a *ServerAuth*. The handle is used in order to associate a *ServerAuth* to an application object. The application must supply a handle when it is informed of the creation of a *ServerAuth*. The SIP Server will supply this handle when calling event functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipServerAuthObjAppHandle);
```

STRUCTURES AND ENUMERATIONS

The Structures and Enumerations are:

- RvSipServerAuthState()
- RvSipServerAuthResult()

RvSipServerAuthState()

DESCRIPTION

Represents the state of a *ServerAuth*. The RvSipServerAuthObjStateChange() callback reports that the specific *ServerAuth* state has changed.

SYNTAX

```
typedef enum{
    RVSIPSERVER_AUTH_STATE_UNDEFINED = -1,
    RVSIPSERVER_AUTH_STATE_IDLE,
    RVSIPSERVER_AUTH_STATE_NEW_AUTH_HEADER,
    RVSIPSERVER_AUTH_STATE_CHECKING_REALM_AND_NONCE,
    RVSIPSERVER_AUTH_STATE_AWAITING_PWD,
    RVSIPSERVER_AUTH_STATE_AUTH_REQUEST,
    RVSIPSERVER_AUTH_STATE_AWAITING_AUTH_HEADER,
    RVSIPSERVER_AUTH_STATE_AUTH_TERMINATE
}RvSipServerAuthState;
```

PARAMETERS

RVSIPSERVER_AUTH_STATE_IDLE

The IDLE state is the initial state of the *ServerAuth*. Upon creation of a *ServerAuth*, the object assumes the IDLE state. It remains in this state until the authentication begins.

RVSIPSERVER_AUTH_STATE_NEW_AUTH_HEADER

The *ServerAuth* object assumes the NEW_AUTH_HEADER state when it starts to authenticate a given Authorization header.

RVSIPSERVER_AUTH_STATE_CHECKING_REALM_AND_NONCE

When the *ServerAuth* finds an Authorization header that uses the Digest scheme with the MD5 algorithm, it assume the CHECKING_REALM_AND_NONCE state. In this state, the *ServerAuth* asks the Security Component whether the Realm and Nonce are valid using the

RvSipServerSecurityAuthCheckRealmAndNonce() interface function. This is an informative state in which the Security Component is not expected to do anything.

RVSIPSERVER_AUTH_STATE_AWAITING_PWD

The *ServerAuth* assumes the AWAITING_PWD state after the realm and nonce were confirmed (digest/MD5). In this state, the Security Component needs to return the password of the related username from the Authorization header of the *ServerAuth*, using RvSipServerAuthObjSetPwd().

RVSIPSERVER_AUTH_STATE_AUTH_REQUEST

The *ServerAuth* assumes the AUTH_REQUEST state when receiving an unknown authentication header, scheme + algorithm (other than digest + MD5). In this state, the Security Component can try to authenticate the given header using any other algorithm known to it. When the Security Component finishes the authentication process, it should call RvSipServerAuthObjSetAuthResult() with the authentication process result.

RVSIPSERVER_AUTH_STATE_AWAITING_AUTH_HEADER

The *ServerAuth* assumes the AWAITING_AUTH_HEADER state after the request authentication failed and the application tried to reject the request with a 401/407 response message (only if the scheme/algorithm is unknown). In this state, the Security Component should build an authentication header, initialize its challenge, and set it in the *ServerAuth* in order to proceed with the response sending process using RvSipServerAuthObjSetAuthenticationHeader().

RVSIPSERVER_AUTH_STATE_AUTH_TERMINATE

This is the final state of the *ServerAuth*. When a *ServerAuth* is terminated, it assumes the TERMINATED state. Upon reaching the TERMINATED state, the application can no longer refer to the *ServerAuth*.

RvSipServerAuthResult()

DESCRIPTION

Represents a *ServerAuth* authentication result. When the *ServerAuth* object assumes the RVSIPSERVER_AUTH_STATE_AUTH_REQUEST state, the Security Component should try to authenticate the authorization header that is in the *ServerAuth*.

The Security Component authentication result should be set to the *ServerAuth*, using RvSipServerAuthObjSetAuthResult(), in order to proceed with the authentication process according to the result of the Security Component.

SYNTAX

```
typedef enum{
    RVSIPSERVER_SECURITY_AUTH_RES_UNDEFINED = -1,
    RVSIPSERVER_SECURITY_AUTH_RES_SUCCESS,
    RVSIPSERVER_SECURITY_AUTH_RES_FAILURE,
    RVSIPSERVER_SECURITY_AUTH_RES_NOT_RELEVANT,
    RVSIPSERVER_SECURITY_AUTH_RES_ERROR,
    RVSIPSERVER_SECURITY_AUTH_RES_STOP
}RvSipServerSecurityAuthResult;
```

PARAMETERS

RVSIPSERVER_AUTH_RES_SUCCESS

The authentication succeeded, the credential was found in the Authorization header of the *ServerAuth*. The authentication processes is done and the associated authenticating object will change its state to REQUEST_VALID.

RVSIPSERVER_AUTH_RES_FAILURE

The authentication failed, the credential was not found in the Authorization header of the *ServerAuth*. The authentication process is done and the associated authenticating object will change its state to REQUEST_INVALID.

RVSIPSERVER_AUTH_RES_NOT_RELEVANT

The current Authorization header is not relevant to this server. Continues with the authentication process and looks for the next Authorization header.

RVSIPSERVER_AUTH_RES_ERROR

An error has occurred while trying to check the Authorization header credential. Continues with the authentication process and looks for the next Authorization header.

RVSIPSERVER_AUTH_RES_STOP

The Security Component wishes to Stop the authentication process. Informs the associated authenticating object to change its state to AUTHENTICATION_FAILED.

Component Interface Type Definitions for Security Server Components

The type definitions are:

- RvSipServerSecurityMgrHandle()
- RvSipServerSecurityInterface()
- RvSipServerAuthObjCreated()
- RvSipServerAuthObjStateChange()
- RvSipServerSecurityAuthCheckRealmAndNonce()
- RvSipServerSecurityAuthGetMD5Credential()
- RvSipServerSecurityAuthMD5()
- RvSipServerSecurityAuthClientSharedSecret()

RvSipServerSecurityMgrHandle()

DESCRIPTION

The handle to the *SecurityMgr* object that the application supplied. This handle will be used for calling the interface functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipServerSecurityMgrHandle);
```

RvSipServerSecurityInterface()

The structure with function pointers to the Security Component interface. This structure is used to set the interface functions to the *SipServerMgr* using RvSipServerAuthMgrSecuritySetInterface().

SYNTAX

```
typedef struct{
    RvSipServerAuthObjCreated
                pfnSecurityServerAuthObjCreated;
    RvSipServerAuthObjStateChange
                pfnSecurityServerAuthStateChange;
    RvSipServerSecurityAuthCheckRealmAndNonce
                pfnSecurityAuthCheckRealmAndNonce;
    RvSipServerSecurityAuthGetMD5Credential
                pfnSecurityServerAuthGetMD5Credential;
    RvSipServerSecurityAuthMD5
                pfnSecurityAuthMD5;
    RvSipServerSecurityAuthClientSharedSecret
                pfnSecurityClientAuthSharedSecret;
    RvSipServerAuthObjCheckRealmAndNonce
                pfnSecurityServerAuthObjCheckRealmAndNonce
}RvSipServerSecurityInterface;
```

PARAMETERS

pfnSecurityAuthStateChange

Notifies that the state of the *ServerAuth* has changed.

pfnSecurityAuthCheckRealmAndNonce

Notifies that the realm and nonce parameters should be checked.

pfnSecurityAuthCreated

Notifies that a new *ServerAuth* object has been created.

pfnSecurityAuthGetMD5Credential

Notifies that MD5 credential are required to proceed with the authentication.

pfnSecurityAuthMD5Ev

Notifies that the MD5 algorithm should be used.

pfnSecurityClientAuthSharedSecret

Notifies that a username and password are needed for client-side authentication.

pfnSecurityServerAuthObjCheckRealmAndNonce

Notifies that the realm and nonce parameters should be checked.

RvSipServerAuthObjCreated()

DESCRIPTION

Notifies the Security Component that a new *ServerAuth* was created. The new *ServerAuth* always assumes the IDLE state. The application can supply a handle to associate with the new *ServerAuth*. This handle is supplied in any of the callbacks relevant to this object.

Note It is not permitted to call the Terminate() API function from this callback.

SYNTAX

```
typedef void (*RvSipServerAuthObjCreated)(
    IN  RvSipServerAuthObjHandle       hServerAuthObj,
    OUT RvSipServerAuthObjAppHandle    *pAppObj);
```

PARAMETERS

hServerAuthObj

The newly created *ServerAuth* object.

pAppObj

The returned application handle for this object.

RETURN VALUES

None.

RvSipServerAuthObjStateChange()

DESCRIPTION

The main *ServerAuth* event. Through this event the Security Component receives a notification that the state of a specific *ServerAuth* has changed. The application can act according to the new object state. For example, when the *ServerAuth* changes its state to AWAITING_PWD, the Security Component should look for a password related to the user name in the Authorization header. After the Security Component finishes looking for the password, it should call RvSipServerAuthObjSetPwd() and report the result to the *ServerAuth*, which will continue the authentication process according to the Security Component result.

SYNTAX

```
typedef void (* RvSipServerAuthObjStateChange)(
    IN RvProxyServerAuthObjHandle       hServerAuthObj,
    IN RvProxyServerAuthObjAppHandle    hAppObj,
    IN RvProxyCoreServerAuthState       eState);
```

PARAMETERS

hServerAuthObj

The *ServerAuth*.

hAppObj

The application handle for this object.

eState

The new *ServerAuth* state.

RETURN VALUES

None.

RvSipServerSecurityAuthCheckRealmAndNonce()

DESCRIPTION

Notifies the Security Component that it must perform realm and nonce checking and indicates whether or not the realm is known to the component. If the realm is known to the component, indicates whether or not the nonce related to this realm is valid.

SYNTAX

```
typedef RvBool(*RvSipServerSecurityAuthCheckRealmAndNonce)(
    IN RvSipServerSecurityMgrHandle    hSecurityMgr,
    IN RvSipMsgHandle                  hMsg,
    IN RPOOL_Ptr                       *pRealm,
    IN RPOOL_Ptr                       *pNonce);
```

PARAMETERS

hSecurityMgr

The *SecurityMgr* handle.

hMsg

The received request.

pRealm

The realm to check.

pNonce

The nonce to check.

RETURN VALUES

RV_TRUE if the realm and nonce are valid. RV_FALSE if either the realm or nonce are invalid.

RvSipServerSecurityAuthGetMD5Credential()

DESCRIPTION

Notifies the Security Component that MD5 credentials are needed in order to continue with the authentication process.

SYNTAX

```
typedef RvStatus(*RvSipServerSecurityAuthGetMD5Credential)(
    IN  RvSipServerAuthObjHandle      hServerAuthObj,
    IN  RvSipServerAuthObjAppHandle   hAppObj,
    IN  RPOOL_Ptr                     *pRequestRealm,
    OUT RPOOL_Ptr                     *pServerRealm,
    OUT RPOOL_Ptr                     *pNonce,
    OUT RPOOL_Ptr                     *pQop,
    OUT RPOOL_Ptr                     *pOpaque,
    OUT RPOOL_Ptr                     *pDomain);
```

PARAMETERS

hServerAuthObj

The *ServerAuth*.

hAppObj

The application owner of the *ServerAuth*.

pRequestRealm

The realm value found on the request. If the request did not contain a realm, the offset of this parameter is UNDEFINED.

pServerRealm

The alternate realm value to use instead of the request realm value.

pNonce

The returned nonce that is related to the realm.

pQop

The returned Qop value.

pOpaque

The returned opaque value.

pDomain

The returned domain value.

Return Values

Returns RvStatus.

RvSipServerSecurityAuthMD5()

DESCRIPTION

Notifies the Security Component that it needs to make a MD5 algorithm calculation.

SYNTAX

```
typedef void (*RvSipServerSecurityAuthMD5)(
    IN  RPOOL_Ptr     *pRpoolMD5Input,
    IN  RvUint32      length,
    OUT RPOOL_Ptr     *pRpoolMD5Output);
```

PARAMETERS

pRpoolMD5Input

The Rpool pointer to the MD5 input.

length

The length of the string inside the page.

strMd5Output

The output of the hash algorithm.

RETURN VALUES

None.

RvSipServerSecurityAuthClientSharedSecret()

DESCRIPTION

Notifies the Security Component that it needs to supply the username and password of the server for client-side authentication.

SYNTAX

```
typedef void (*RvSipServerSecurityAuthClientSharedSecret)(
    IN  RvSipAuthenticatorHandle       hAuthenticator,
    IN  RvSipAppAuthenticatorHandle    hAppAuthenticator,
    IN  void*                          notUsed1,
    IN  void*                          notUsed2,
    IN  RPOOL_Ptr                      *pRpoolRealm,
    OUT RPOOL_Ptr                      *pRpoolUserName,
    OUT RPOOL_Ptr                      *pRpoolPassword);
```

PARAMETERS

hAuthenticator

The handle to the authenticator object.

hAppAuthenticator

The handle to the application authenticator handle.

notUsed1

Always NULL. Not used in this version.

notUsed2

Always NULL. Not used in this version.

pRpoolRealm

The realm string in RPool_ptr format.

pRpoolUserName

The username string in RPool_ptr format.

pRpoolPassword

The password string in RPool_ptr format.

RETURN VALUES

None.

16

TRANSPORT API TYPE DEFINITIONS

WHAT'S IN THIS CHAPTER

This section includes the SIP Stack Transport type definitions and callback functions defined in the *RvSipTransportTypes.h* and *RvSipTransportDNSTypes.h* header files.

This section includes:

HANDLE TYPE DEFINITIONS

The Handle type definitions are:

- RvSipTransportMgrHandle
- RvSipTransportConnectionHandle
- RvSipTransportConnectionOwnerHandle
- RvSipTransportConnectionAppHandle
- RvSipTransportTlsEngineHandle
- RvSipTransportDNSListHandle
- RvSipTransportLocalAddrHandle
- RvSipTransportTlsCertificate
- RvSipServerTransportMgrHandle
- RvSipServerTransportMgrAppHandle

RvSipTransportMgrHandle

DESCRIPTION

The declaration of a handle to the *TransportMgr*.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipTransportMgrHandle);
```

PARAMETERS

None.

RETURN VALUES

None.

RvSipTransportConnectionHandle

DESCRIPTION

The declaration of handle to a connection object. A connection object is used for TCP/TLS communication.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipTransportConnectionHandle);
```

PARAMETERS

None.

RETURN VALUES

None.

RvSipTransportConnectionOwnerHandle

DESCRIPTION

The declaration of a handle to the owner of a connection object. A connection can have several owners. All owners are notified of connection events.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipTransportConnectionOwnerHandle);
```

PARAMETERS

None.

RETURN VALUES

None.

RvSipTransportConnectionAppHandle

DESCRIPTION

An application handle that the application can set and get from the connection. A connection can hold only one application handle.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipTransportConnectionAppHandle);
```

PARAMETERS

None.

RETURN VALUES

None.

RvSipTransportTlsEngineHandle

DESCRIPTION

The declaration of handle to a transport TLS Engine. The SIP Stack can hold several TLS engines, each with different attributes. The TLS engines can be used to set different sets of TLS attributes to different connections.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipTransportTlsEngineHandle);
```

PARAMETERS

None.

RETURN VALUES

None.

RvSipTransportDNSListHandle

DESCRIPTION

The deceleration of handle to a DNS list handle. The DNS list is used as an interface between the SIP Stack DNS lists and the application. In this list the SIP Stack holds SRV host and IP records.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipTransportDNSListHandle);
```

PARAMETERS

None.

RETURN VALUES

None.

RvSipTransportLocalAddrHandle

DESCRIPTION

The handle to the local address object, which the SIP Stack uses for receiving or sending messages, and for listening for new connections.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipTransportLocalAddrHandle);
```

RvSipTransportTlsCertificate

DESCRIPTION

A pointer to a certificate retrieved from a connection or used in the compare certificate callback

SYNTAX

```
RV_DECLARE_HANDLE(RvSipTransportTlsCertificate);
```

PARAMETERS

None.

RETURN VALUES

None.

The handles are:

- RvSipServerTransportMgrHandle
- RvSipServerTransportMgrAppHandle

RvSipServerTransportMgrHandle

DESCRIPTION

The declaration of a handle to the *ServerTransportMgr*. The *ServerTransportMgr* is responsible for registering the application implementation on SIP Stack transport events.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipServerTransportMgrHandle);
```

RvSipServerTransportMgrAppHandle

DESCRIPTION

The declaration of a handle to an application *ServerTransportMgr*.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipServerTransportMgrAppHandle);
```

Structures and Enumerations

The structures and enumerations are:

- RvSipTransportAddressType
- RvSipTransportPersistencyLevel
- RvSipTransportConnectionCfg
- RvSipTransportConnectionState
- RvSipTransportConnectionTlsState
- RvSipTransportConnectionStatus
- RvSipTransportConnectionTlsStatus
- RvSipTransportConnectionStateChangedReason
- RvSipTransportPrivateKeyType
- RvSipTransportTlsMethod
- RvSipTransportMsgAddrCfg
- RvSipTransportBsAction
- RvSipTransportMgrEvHandlers
- RvSipTransportConnectionEvHandlers
- RvSipTransportTlsEngineCfg
- RvSipTransportTlsHandshakeSide
- RvSipTransportAddr
- RvSipTransportOutboundProxyCfg
- RvSipTransportAddrOptions
- RvSipTransportOutboundAddress
- RvSipServerTransportEvHandlers
- RvSipServerTransportLocalInterfaceRecord
- RvSipTransportRcvdMsgDetails
- RvSipTransportConnectionType

RvSipTransportAddressType

DESCRIPTION

Specifies whether an address is an IPv4 or an IPv6 address.

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_ADDRESS_TYPE_UNDEFINED = -1,
    RVSIP_TRANSPORT_ADDRESS_TYPE_IP,
    RVSIP_TRANSPORT_ADDRESS_TYPE_IP6
}RvSipTransportAddressType;
```

RvSipTransportPersistencyLevel

DESCRIPTION

Defines the persistency level in the system.

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_PERSISTENCY_LEVEL_UNDEFINED = -1,
    RVSIP_TRANSPORT_PERSISTENCY_LEVEL_TRANSC,
    RVSIP_TRANSPORT_PERSISTENCY_LEVEL_TRANSC_USER
}RvSipTransportPersistencyLevel;
```

PARAMETERS

RVSIP_TRANSPORT_PERSISTENCY_LEVEL_UNDEFINED

When the SIP Stack is configured to use an undefined persistency level, the following rules apply:

- SIP Stack objects do not look for suitable connections in the hash before sending a message, and therefore always open new connections for sending requests. (Responses are still sent on the connection on which the request was received).
- Newly created connections are not inserted into the hash.

RVSIP_TRANSPORT_PERSISTENCY_LEVEL_TRANSC

When the SIP Stack is configured to use the Transaction persistency level, the following rules apply:

- A transaction object that wishes to send a request will first try to locate a suitable connection in the hash.
- If there is a suitable open connection, the transaction will use it. If there is not, the transaction will open a new connection and insert it into the connections hash. In both cases, the transaction will attach itself to the connection and become the connection owner.
- The transaction will detach from the connection only before the transaction terminates.

RVSIP_TRANSPORT_PERSISTENCY_LEVEL_TRANSC_USER

A Transaction User (TU) is an object that uses transaction objects for sending requests. A call-leg, subscription and a register-client are all Transaction Users. When the SIP Stack is configured to use the TU persistency level, the following rules apply:

- SIP Stack transactions behave as defined in the Transaction persistency level.
- A TU tries to use the same connection for all outgoing requests (sent by different client transactions).

RvSipTransportConnectionCfg

DESCRIPTION

A structure containing the configuration needed to initialize a new connection. You need to supply this structure when calling the RvSipTransportConnectionInit() function.

SYNTAX

```
typedef struct{
    IN RvSipTransport    eTransportType;
    IN RvChar*           strLocalIp;
    IN RvUint16          localPort;
    IN RvChar*           strDestIp;
    IN RvUint16          destPort;
    IN RvChar*           strHostName;
}RvSipTransportConnectionCfg;
```

PARAMETERS

eTransportType

The transport type—TCP or TLS.

strLocalIp

The local IP as a string. IPv6 should be in []%sid format. If null is supplied, a default local IP is chosen.

localPort

The local port. 0 is replaced with the transport default port (5060 for TCP and 5061 for TLS). This parameter is ignored if *strLocalIp* is not specified.

strDestIp

The destination IP as a string. IPv6 should be supplied in a [] format.

strDestPort

The destination port. 0 is replaced with the transport default port (5060 for TCP and 5061 for TLS).

bEnablePersistency

Specifies whether or not to insert the connection to the hash.

strHostName

If the connection type is TLS, this string will be used for post connection assertion

RvSipTransportConnectionState

DESCRIPTION

Represents the state of a connection. The state changes are reported using the RvSipServerTransportConnectionStateChangedEv() callback function with a state change reason.

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_CONN_STATE_UNDEFINED = -1,
    RVSIP_TRANSPORT_CONN_STATE_IDLE,
    RVSIP_TRANSPORT_CONN_STATE_READY,
    RVSIP_TRANSPORT_CONN_STATE_CONNECTING,
    RVSIP_TRANSPORT_CONN_STATE_TCP_CONNECTED,
    RVSIP_TRANSPORT_CONN_STATE_CLOSING,
    RVSIP_TRANSPORT_CONN_STATE_CLOSED,
    RVSIP_TRANSPORT_CONN_STATE_TERMINATED
}RvSipTransportConnectionState;
```

PARAMETERS

RVSIP_TRANSPORT_CONN_STATE_IDLE

The IDLE state is the initial state of the Connection state machine. Upon creation of the connection, the connection assumes the IDLE state. It remains in this state until the RvSipTransportInit() function is called, whereupon it should move to the READY state.

RVSIP_TRANSPORT_CONN_STATE_READY

After calling the RvSipTransportInit() function that initializes the connection and inserts it into the connection hash, the connection enters the READY state. The connection will move from the READY state to the CONNECTING state in the following cases:

- If the application specifically called the RvSipTransportInitConnect() function on this connection.

- If one of the persistent SIP Stack objects located this connection in the hash and connected it to send a message. Calling the RSipTransportTerminate() function in the READY state will move the connection to the TERMINATED state.

RVSIP_TRANSPORT_CONN_STATE_CONNECTING

Calling the RvSipTransporConnect() function in the READY state will move the connection to the CONNECTING state. While in the CONNECTING state, the SIP Stack tries to connect the connection to the remote party. If the connect attempt succeeds, the connection assumes the CONNECTED state. Otherwise, the connection assumes the TERMINATED state with the RVSIP_TRANSPORT_CONN_REASON_ERROR reason.

RVSIP_TRANSPORT_CONN_STATE_TCP_CONNECTED

Upon receiving the network connected event, the connection moves to the CONNECTED state. While in this state, the connection can send and receive SIP messages. Calling RvSipTransportTerminate() in this state will move the connection to the CLOSING state and the connection will start the disconnection procedure. This will also be the case for client connections that are left with no owners.

RVSIP_TRANSPORT_CONN_STATE_CLOSING

A client connection that remains with no owners, or any connection on which the application called the RvSipTransportTerminate() function moves to the CLOSING state. While in this state, the connection cleans and then shuts down the established socket connection. The connection will move to the CLOSED state upon receiving a network event that indicates that the connection was closed by the remote party.

RVSIP_TRANSPORT_CONN_STATE_CLOSED

The connection moves to the CLOSED state upon receiving a network event that indicates that the connection was closed by the remote party. In the CLOSED state, the connection closes its internal socket. The connection then moves to the TERMINATED state.

RVSIP_TRANSPORT_CONN_STATE_TERMINATED

The final connection state. When a connection is terminated, the connection assumes the TERMINATED state. Upon reaching the TERMINATED state, you can no longer reference the connection object.

RvSipTransportConnectionTlsState

DESCRIPTION

The TLS state of the connection.

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_CONN_TLS_STATE_UNDEFINED = -1,
    RVSIP_TRANSPORT_CONN_TLS_STATE_HANDSHAKE_COMPLETED,
    RVSIP_TRANSPORT_CONN_TLS_STATE_HANDSHAKE_READY,
    RVSIP_TRANSPORT_CONN_TLS_STATE_HANDSHAKE_STARTED,
    RVSIP_TRANSPORT_CONN_TLS_STATE_HANDSHAKE_FAILED,
    RVSIP_TRANSPORT_CONN_TLS_STATE_CLOSE_SEQUENSE_STARTED,
    RVSIP_TRANSPORT_CONN_TLS_STATE_CONNECTED,
    RVSIP_TRANSPORT_CONN_TLS_STATE_TERMINATED
}RvSipTransportConnectionTlsState;
```

PARAMETERS

RVSIP_TRANSPORT_CONN_TLS_STATE_UNDEFINED

No TLS sequence was initiated on the connection.

RVSIP_TRANSPORT_CONN_TLS_STATE_HANDSHAKE_COMPLETED

The handshake procedure on the connection was completed.

RVSIP_TRANSPORT_CONN_TLS_STATE_HANDSHAKE_READY

The connection is TCP-connected and ready to start the TLS handshake.

RVSIP_TRANSPORT_CONN_TLS_STATE_HANDSHAKE_STARTED

The connection is performing the TLS handshake.

RVSIP_TRANSPORT_CONN_TLS_STATE_HANDSHAKE_FAILED

The TLS handshake failed. No data can be transmitted on the connection.

RVSIP_TRANSPORT_CONN_TLS_STATE_CLOSE_SEQUENSE_STARTED

The connection has received or sent a close request but is not yet closed.

RVSIP_TRANSPORT_CONN_TLS_STATE_CONNECTED

Data can be sent on the connection.

RVSIP_TRANSPORT_CONN_TLS_STATE_TERMINATED

The connection is terminated. After this point, the connection cannot be accessed again.

RvSipTransportConnectionStatus

DESCRIPTION

Connection events that do not effect the connection state.

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_CONN_STATUS_UNDEFINED = 1,
    RVSIP_TRANSPORT_CONN_STATUS_ERROR,
    RVSIP_TRANSPORT_CONN_STATUS_MSG_SENT,
    RVSIP_TRANSPORT_CONN_STATUS_MSG__NOT_SENT
}RvSipTransportConnectionStatus;
```

PARAMETERS

RVSIP_TRANSPORT_CONN_STATUS_UNDEFINED

Unspecified status.

RVSIP_TRANSPORT_CONN_STATUS_ERROR

An error occurred. The connection will be closed.

RVSIP_TRANSPORT_CONN_STATUS_MSG_SENT

For internal use only. The application will not be notified of this status.

RVSIP_TRANSPORT_CONN_STATUS_MSG_NOT_SENT

A message that was supposed to be sent by this connection was not sent and the connection was closed. Only the owner of this message is notified about this status.

RvSipTransportConnectionTlsStatus

DESCRIPTION

The status of a connection.

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_CONN_TLS_STATUS_UNDEFINED = 1,
    RVSIP_TRANSPORT_CONN_TLS_STATUS_HANDSHAKE_PROGRESS
}RvSipTransportConnectionTlsStatus;
```

PARAMETERS

RVSIP_TRANSPORT_CONN_TLS_STATUS_UNDEFINED

No security status.

RVSIP_TRANSPORT_CONN_TLS_STATUS_HANDSHAKE_PROGRESS

Data was sent or received during a handshake process.

RvSipTransportConnectionStateChangedReason

DESCRIPTION

The reason supplied with the state changed function of a connection. The reason is used only if it supplies more information about the new state. Otherwise, the reason is undefined.

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_CONN_REASON_UNDEFINED = -1,
    RVSIP_TRANSPORT_CONN_REASON_ERROR,
    RVSIP_TRANSPORT_CONN_REASON_CLIENT_CONNECTED,
    RVSIP_TRANSPORT_CONN_REASON_SERVER_CONNECTED,
    RVSIP_TRANSPORT_CONN_REASON_TLS_POST_CONNECTION_ASSERTIO
N_FAILED,
    RVSIP_TRANSPORT_CONN_REASON_DISCONNECTED
}RvSipTransportConnectionStateChangedReason;
```

RvSipTransportPrivateKeyType

DESCRIPTION

Determines the key type to use in a TLS engine.

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_PRIVATE_KEY_TYPE_UNDEFINED = -1,
    RVSIP_TRANSPORT_PRIVATE_KEY_TYPE_RSA_KEY
}RvSipTransportPrivateKeyType;
```

RvSipTransportTlsMethod

DESCRIPTION

Determines the version of TLS to use in an engine.

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_TLS_METHOD_UNDEFINED = -1,
    RVSIP_TRANSPORT_TLS_METHOD_SSL_V2,
    RVSIP_TRANSPORT_TLS_METHOD_SSL_V3,
    RVSIP_TRANSPORT_TLS_METHOD_TLS_V1
}RvSipTransportTlsMethod;
```

RvSipTransportMsgAddrCfg

DESCRIPTION

A structure containing the configuration needed to inject a message into the SIP Stack. You need to supply this structure when calling the RvSipTransportInjectMsg() function. When injecting a message into the SIP Stack, the SIP Stack behaves as if the message was received from the network. Therefore, you need to supply the addresses (local and remote) that you want to use for this message with the injected message. These addresses are included in the RvSipTransportMsgAddrCfg structure.Using this structure you can simulate that a message was received from a certain destination and on a specific local address. The local address will also be used if a response is sent to an incoming injected request.

SYNTAX

```
typedef struct{
    IN RvSipTransport    eTransportType;
    IN RvChar*           strLocalIp;
    IN RvUint16          localPort;
    IN RvChar*           strDestIp;
    IN RvUint16          destPort;
}RvSipTransportMsgAddrCfg;
```

PARAMETERS

eTransportType

The transport type—UDP, TCP or TLS. If UUNDEFINED, use the default UDP transport type.

strLocalIp

The local IP as a string. IPv6 should be in []%sid format. If NULL is supplied, a default local IP is chosen.

localPort

The local port. 0 is replaced with the transport default port (5060 for UDP and TCP, 5061 for TLS).

strDestIp

The destination IP as a string. IPv6 should be supplied in [] format. If NULL, no destination address will be set, and a response will be sent according to the address in the top Via header.

strDestPort

The destination port. 0 is replaced with the transport default port (5060 for UDP and TCP, 5061 for TLS).

RvSipTransportBsAction

DESCRIPTION

Determines how to proceed with handling a received message with bad syntax. This definition is for use as an output parameter, in the RvSipServerTransportBadSyntaxMsgEv() and RvSipServerTransportBadSyntaxStartLineMsgEv() callback functions.

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_BS_ACTION_UNDEFINED = -1,
    RVSIP_TRANSPORT_BS_ACTION_DISCARD_MSG,
    RVSIP_TRANSPORT_BS_ACTION_REJECT_MSG,
    RVSIP_TRANSPORT_BS_ACTION_CONTINUE_PROCESS
}RvSipTransportBsAction;
```

PARAMETERS

RVSIP_TRANSPORT_BS_ACTION_DISCARD_MSG

Does nothing, discard the bad-syntax message. (The same behavior as was in SIP Stack version 2.2.)

RVSIP_TRANSPORT_BS_ACTION_REJECT_MS

Sends a 400 response for the bad-syntax message. (Relevant only for Request messages.)

RVSIP_TRANSPORT_BS_ACTION_CONTINUE_PROCESS

The SIP Stack continues with message processing, as much as possible. The SIP Stack will process the bad-syntax message until it finds that an essential header has a syntax error, and then send a 400 response. If all the essential headers are correct, the SIP Stack will successfully process the message.

RvSipTransportMgrEvHandlers

DESCRIPTION

A structure with function pointers to the Transport module callbacks. This structure is used to set the application callbacks in the RvSipTransportMgrSetEvHandlers() function.

SYNTAX

```
typedef struct{
    RvSipTransportMsgToSendEv
                            pfnEvMsgToSend;
    RvSipTransportMsgReceivedEv
                            pfnEvMsgRecvd;
    RvSipTransportMsgThreadError
                            pfnThreadError;
    RvSipTransportBadSyntaxMsgEv
                            pfnEvBadSyntaxMsg;
    RvSipTransportBadSyntaxStartLineMsgEv
                            pfnEvBadSyntaxStartLineMsg;
    RvSipTransportConnectionTlsStateChangedEv
                            pfnEvTlsStateChanged;
    RvSipTransportConnectionTlsPostConnectionAssertionEv
                            pfnEvTlsPostConnectionAssertion;
    RvSipTransportConnectionTlsSequenceStartedEv
                            pfnEvTlsSeqStarted;
    RvSipTransportBufferReceivedEv
                            pfnEvBufferReceived;
    RvSipTransportBufferToSendEv
                            pfnEvBufferToSend;
    RvSipTransportConnectionCreatedEv
                            pfnEvConnCreated;
    RvSipTransportConnectionParserResultEv
                            pfnEvConnParserResult;
    RvSipTransportConnectionStateChangedEv
                            pfnEvConnStateChanged;
```

```
#ifdef (RV_SSL_SESSION_STATUS)
    RvSipTransportConnectionTlsStatusEv
                                pfnEvConnTlsStatus;
#endif
}RvSipTransportMgrEvHandlers;
```

PARAMETERS

pfnEvMsgToSend

For internal use only.

pfnEvMsgReceived

For internal use only.

pfnThreadError

A thread that was exited unexpectedly.

pfnEvBadSyntaxMsg

Notifies that a bad-syntax message was received.

pfnEvBadSyntaxStartLineMsg

Notifies that a message with a bad-syntax start-line was received.

pfnEvTlsStateChanged

Notifies the application on TLS connection state changes.

pfnEvTlsPostConnectionAssertion

If set to something other the NULL, this parameter will let the application override post connection assertions that fail.

pfnEvTlsSeqStarted

The beginning of a TLS procedure on a connection.

pfnEvBufferReceived

Gives the application the opportunity to discard the buffer before parsing. The remote and local addresses are supplied. The application can also dump the buffer.

pfnEvBufferToSend

Allows the application to dump the message buffer.

pfnEvConnCreated

Notifies application about the creation of connection object for an incoming TCP connection.

pfnEvConnParserResult

The result of the parsing of incoming message.

pfnEvConnStateChanged

Notifies the application about an incoming Connection state change.

pfnEvConnTlsStatus

Notifies the application about security-related issues using the security callback.

RvSipTransportConnectionEvHandlers

DESCRIPTION

A structure with function pointers to the connection callbacks. You should supply this structure for every new connection.

SYNTAX

```
typedef struct{
    RvSipTransportConnectionStateChangedEv
                              pfnConnStateChangedEvHandler;
    RvSipTransportConnectionStatusEv
                              pfnConnStausEvHandler;
    }RvSipTransportConnectionEvHandlers;
```

PARAMETERS

pfnConnStateChangedEvHandler

A callback function to notify the owner about connection states.

pfnConnStausEvHandler

A callback function to notify the owner about connection statuses, such as ERROR.

RvSipTransportTlsEngineCfg

A structure containing the configuration needed to initialize a new TLS engine. You need to supply this structure when calling the RvSipTransportTlsEngineConstruct() function.

SYNTAX

```
typedef struct{
    RvSipTransportTlsMethod         eTlsMethod;
    RvChar*                         strPrivateKey;
    RvSipTransportPrivateKeyType    ePrivateKeyType;
    RvInt32                         privateKeyLen;
    RvChar*                         strCert;
    RvInt32                         certLen;
    RvInt32                         certDepth;
}RvSipTransportTlsEngineCfg;
```

PARAMETERS

eTlsMethod

The SSL/TLS version.

strPrivateKey

The private key for the engine.

ePrivateKeyType

The private key type.

privateKeyLen

The size of the key.

strCert

The certificate issued for the engine.

certLen

The size of the certificate.

certDepth

The maximum length of a chain of certificate before it is considered invalid.

RvSipTransportTlsHandshakeSide

DESCRIPTION

Determines the side of the TLS handshake to assume. The default means that the TCP client will act as a TLS client and the TCP server will act as a TLS server. The default is the recommended way of working.

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_TLS_HANDSHAKE_SIDE_UNDEFINED = -1,
    RVSIP_TRANSPORT_TLS_HANDSHAKE_SIDE_DEFAULT,
    RVSIP_TRANSPORT_TLS_HANDSHAKE_SIDE_CLIENT,
    RVSIP_TRANSPORT_TLS_HANDSHAKE_SIDE_SERVER
}RvSipTransportTlsHandshakeSide;
```

RvSipTransportAddr

DESCRIPTION

A structure that contains parameters of the address that the SIP Stack can use for network communication.

SYNTAX

```
typedef struct{
    RvSipTransport              eTransportType;
    RvSipTransportAddressType   eAddrType;
    RvUint16                    port;
    RvChar                      strIP;
    RvInt                       Ipv6Scope;
}RvSipTransportAddr;
```

PARAMETERS

eTransportType

The type of the transport protocol that is used with the address.

eAddrType

The type of IP address, such as IPv4 and IPv6.

port

The port.

strIp [RVSIP_TRANSPORT_LEN_STRING_IP]

A NULL terminated string that represents the IP. (If the address is an IPv6 address, the string should be in the following format: xxxx:xxxx:...:xxxx).

Ipv6Scope

The scope of the IPv6 address if it is local.

RvSipTransportOutboundProxyCfg

DESCRIPTION

A structure containing all configurations that are needed to set the outbound proxy details.

SYNTAX

```
typedef struct{
    RvChar           *strIpAddress;
    RvInt32           port;
    RvChar            *strHostName;
    RvSipTransport    eTransport;
#ifdef RV_SIGCOMP_ON
    RvSipCompType     eCompression;
#endif
}RvSipTransportOutboundProxyCfg;
```

PARAMETERS

strIpAddress

The IP address of an outbound proxy that the SIP Stack uses.

Default Value: NULL—no outbound proxy.

port

The port of the outbound proxy that the SIP Stack uses.

Default Value: 5060

strHostName

The host name of an outbound proxy that the SIP Stack uses. For each outgoing request, the DNS will be queried for this host IP address.

Remark: If you set the *outboundProxyIPAddress* parameter, the *outboundProxyHostName* parameter will be ignored.

Default value: NULL

eTransport

The transport of the outbound proxy that the SIP Stack uses.

Default Value: RVSIP_TRANSPORT_UNDEFINED

eCompression

The compression type of the SIP Stack messages that are sent to the outbound proxy.

Default Value: RVSIP_COMP_UNDEFINED

RvSipTransportAddrOptions

DESCRIPTION

A structure containing some parameters of the address.

SYNTAX

```
typedef struct{
    RvSipCompType    eCompression;
}RvSipTransportAddrOptions;
```

PARAMETERS

eCompression

The type of compression that the address uses.

RvSipTransportOutboundAddress

DESCRIPTION

A structure containing the configuration needed to set the outbound server details which will be used for outgoing messages that are sent toward a server.

SYNTAX

```
typedef struct{
    RV_CHAR*          strIpAddress;
    RV_INT32          port;
    RV_CHAR*          strHostName;
    RvSipTransport    eTransport;
}RvSipTransportOutboundAddress;
```

PARAMETERS

strIpAddress

The IP address of an outbound proxy that the SIP Stack uses.

Default Value: 0—no outbound proxy

port

The port of the outbound proxy that the SIP Stack uses.

Default Value: 5060

strHostName

The host name of an outbound proxy that the SIP Stack uses. For each outgoing request, the DNS will be queried for this host IP address.

Remark: If you set the *outboundProxyIPAddress* parameter, the *outboundProxyHostName* parameter will be ignored.

Default Value: NULL

eTransport

The transport of the outbound proxy that the SIP Stack uses.

Default Value: RVSIP_TRANSPORT_UNDEFINED

RvSipServerTransportEvHandlers

DESCRIPTION

A structure with function pointers to the callbacks of the Server Transport module. This structure is used to set the callback implementations of the application. To set the callback implementations, the application should call RvSipServerTransportMgrSetEvHandler().

SYNTAX

```
typedef struct{
  RvSipServerTransportConnectionTlsStateChangedEv
                              pfnEvTlsStateChanged;
  RvSipServerTransportConnectionTlsPostConnectionAssertionEv
                              pfnEvTlsPostConnectionAssertion;
  RvSipServerTransportConnectionTlsSequenceStartedEv
                              pfnEvTlsSeqStarted;
  RvSipServerTransportBadSyntaxMsgEv
                              pfnEvBadSyntaxMsg;
  RvSipServerTransportBadSyntaxStartLineMsgEv
                              pfnEvBadSyntaxStartLineMsg;
  RvSipServerTransportConnectionCreatedEv
                              pfnEvConnCreated;
  RvSipServerTransportConnectionStateChangedEv
                              pfnEvConnStateChanged;
  RvSipServerTransportBufferReceivedEv
                              pfnEvBufferReceived;
  RvSipServerTransportBufferToSendEv
                              pfnEvBufferToSend;
  RvSipServerTransportConnectionParserResultEv
                              pfnEvConnParserResult;
  RvSipServerTransportConnectionDataReceivedEv
                              pfnEvConnDataReceived;
  RvSipServerTransportRegExpResolutionNeededEv
                              pfnEvRegExpResolutionNeeded;
  RvSipServerTransportResolveAddressEv
```

```
                                pfnEvResolveAddress;
  RvSipServerTransportConnectionServerReuseEv
                                pfnEvConnServerReuse;
  RvSipServerTransportMsgReceivedEv
                                pfnEvMsgReceived;
}RvSipServerTransportEvHandler;
```

PARAMETERS

pfnEvTlsStateChanged

Notifies the application on TLS connection state changes.

pfnEvTlsPostConnectionAssertion

If set to a value other than NULL, will enable the application to control post connection assertions.

pfnEvTlsSeqStarted

Indicates the beginning of a TLS procedure on a connection.

pfnEvBadSyntaxMsg

Notifies that a bad-syntax message was received.

pfnEvBadSyntaxStartLineMsg

Notifies that a message with a bad-syntax start-line was received.

pfnEvConnCreated

Notifies the application about the creation of a connection object for an incoming TCP connection.

pfnEvConnStateChanged

Notifies the application about an incoming connection state change.

pfnEvBufferReceived

Gives the application the opportunity to discard a buffer before parsing. Remote and local addresses are supplied. The buffer can also be dumped.

pfnEvBufferToSend

Enables dumping of sent messages by the application.

pfnEvConnParserResult

Indicates the result of the parsing of an incoming message.

pfnEvConnDataReceived

Notifies application that data was read from the socket of the incoming connection.

pfnEvRegExpResolutionNeeded

Notifies the application to parse the given unresolved address according to a regular expression retrieved from the DNS server.

pfnEvResolveAddress

Notifies the application to perform resolution by itself.

pfnEvConnServerReuse

Notifies application that a server connection can be reused.

pfnEvMsgReceived

Notifies the application that a new message was received.

RvSipServerTransportLocalInterfaceRecord

DESCRIPTION

The local interface record data structure.

SYNTAX

```
typedef struct{
    RvInt32                                 sizeOfStruct;
    RvSipTransportAddr                      localAddress;
    RvSipAddressHandle                      hRecordRouteAddress;
    RvSipAddressHandle      hResolvedTlsRecordRouteAddress;
    RvSipViaHeaderHandle                     hVia;
    RvSipServerTransportMgrAppHandle        hAppHandle;
}RvSipServerTransportLocalInterfaceRecord;
```

PARAMETERS

sizeOfStruct

The size of this structure.

localAddress

The local address.

hRecordRouteAddress

The Record-Route address.

hResolvedTlsRecordRouteAddress

The Record-Route address which is resolved (by DNS) to the TLS address. This applies only to TLS transport and will be ignored in other cases.

hVia

The Via header.

hAppHandle

The application handle related to this LIR.

RvSipTransportRcvdMsgDetails

DESCRIPTION

A structure containing all the data that is related to a received message.

SYNTAX

```
typedef struct{
    RvSipTransportConnectionHandle    hConnInfo;
    RvSipTransportLocalAddrHandle     hLocalAddr;
    RvSipTransportAddr                recvFromAddr;
    RvSipCompType                     eCompression;
    RvSipTransportBsAction            eBSAction;
}RvSipTransportRcvdMsgDetails;
```

PARAMETERS

hConnInfo

The message *connection* handle or NULL when UDP was used.

hLocalAddr

The local address on which the message was received.

recvFromAddr

The address from where the message was received.

eCompression

The compression type of the message.

eBSAction

A bad-syntax action that was given by application in bad-syntax callbacks. The action may be continue processing, reject or discard.

RvSipTransportConnectionType

DESCRIPTION

The type of the connection-oriented *connection* (for example, TCP, SCTP).

SYNTAX

```
typedef enum{
    RVSIP_TRANSPORT_CONN_TYPE_UNDEFINED,
    RVSIP RVSIP_TRANSPORT_CONN_TYPE_CLIENT,
    RVSIP_TRANSPORT_CONN_TYPE_SERVER,
    RVSIP_TRANSPORT_CONN_TYPE_MULTISERVER,
}RvSipTransportConnectionType;
```

PARAMETERS

RVSIP_TRANSPORT_CONN_TYPE_UNDEFINED

Undefined type.

RVSIP RVSIP_TRANSPORT_CONN_TYPE_CLIENT

Client *connection* (outgoing).

RVSIP_TRANSPORT_CONN_TYPE_SERVER

Server *connection* (incoming).

RVSIP_TRANSPORT_CONN_TYPE_MULTISERVER

Connections holding the socket, on which the Stack listens for incoming *connections*.

TRANSPORT DNS TYPE DEFINITIONS

The Transport DNS type definitions are:

- RvSipTransportDNSSRVElement
- RvSipTransportDNSHostNameElement
- RvSipTransportDNSIPElement

RvSipTransportDNSSRVElement

DESCRIPTION

Defines the external API SRV name element, produced by the NAPTR DNS query and used for the SRV DNS query. According to RFC1035 (DNS), DNS can return a string of a maximum of 255 characters.

SYNTAX

```
typedef struct{
    RvChar            srvName[256];
    RvSipTransport    protocol;
    RvUint16          order;
    RvUint16          preference;
}RvSipTransportDNSSRVElement;
```

PARAMETERS

srvName

The SRV name for the DNS SRV query.

protocol

The transport protocol, discovered by the NAPTR query, or undefined.

order

The NAPTR order, discovered by the NAPTR query, or undefined.

preference

The NAPTR preference, discovered by the NAPTR query, or undefined.

RvSipTransportDNSHostNameElement

DESCRIPTION

Defines the DNS host name list element, which keeps information according to a single host name that may be retrieved by the SRV DNS query. According to RFC1035 (DNS), DNS can return a string of a maximum of 255 characters.

SYNTAX

```
typedef struct{
    RvChar            hostName[256];
    RvSipTransport    protocol;
    RvUint16          port;
    RvUint16          priority;
    RvUint16          weight;
}RvSipTransportDNSHostNameElement;
```

PARAMETERS

hostName

The host name for DNS “A/AAAA” query.

protocol

The transport protocol, discovered by the SRV query, defined explicitly or undefined.

port

The port, discovered by the SRV query, defined explicitly or undefined.

priority

The priority, discovered by the SRV query, or undefined

weight

The weight, discovered by the SRV query, or undefined

RvSipTransportDNSIPElement

DESCRIPTION

Defines the DNS IP address list element, which keeps information according to a single IP address.

SYNTAX

```
typedef struct{
    RvUint8           ip[RVSIP_TRANSPORT_LEN_BYTES_IPV6];
    RvBool            bIsIpV6;
    RvSipTransport    protocol;
    RvUint16          port;
}RvSipTransportDNSIPElement;
```

PARAMETERS

ip

The binary IP address.

bIsIpV6

TRUE if the IP address is the IPv6 address.

protocol

The transport protocol, discovered by the NAPTR query, or undefined

port

The destination port that should be connected.

TRANSPORT CALLBACK FUNCTIONS

The Transport callback functions are:

- RvSipServerTransportConnectionStateChangedEv()
- RvSipServerTransportConnectionTlsSequenceStartedEv()
- RvSipServerTransportConnectionTlsStateChangedEv()
- RvSipServerTransportConnectionTlsPostConnectionAssertionEv()
- RvSipServerTransportBadSyntaxMsgEv()
- RvSipServerTransportBadSyntaxStartLineMsgEv()
- RvSipServerTransportConnectionCreatedEv()
- RvSipServerTransportBufferReceivedEv()
- RvSipServerTransportBufferToSendEv()
- RvSipServerTransportConnectionParserResultEv()
- RvSipServerTransportConnectionDataReceivedEv()
- RvSipServerTransportResolveAddressEv()
- RvSipServerTransportConnectionServerReuseEv()
- RvSipServerTransportRegExpResolutionNeededEv()

RvSipServerTransportConnectionStateChangedEv()

DESCRIPTION

The connection is a stateful object that can assume different states according to the Connection state machine. Through this function, you receive notifications of connection state changes and the associated state change reason.

In a regular connection life cycle, the reason for the state is set to RVSIP_TRANSPORT_CONN_REASON_UNDEFINED. When the connection is closed because of an error, the reason is set to RVSIP_TRANSPORT_CONN_REASON_ERROR.

Note You do not have to register to this callback if you do not want to get connection states.

SYNTAX

```
typedef RvStatus (RVCALLCONV
*RvSipServerTransportConnectionStateChangedEv) (
   IN RvSipTransportConnectionHandle              hConn,
   IN RvSipTransportConnectionOwnerHandle         hObject,
   IN RvSipTransportConnectionState               eState,
   IN RvSipTransportConnectionStateChangedReason  eReason);
```

PARAMETERS

hConn

The connection handle.

hObject

For a server connection, the application handle.

For a client connection, the handle to the connection owner.

eState

The connection state.

eReason

A reason for the new state, or undefined if there is no special reason.

RETURN VALUES

Returns RvStatus. (The returned status is ignored in the current SIP Stack version.)

RvSipServerTransportConnectionTlsSequenceStartedEv()

DESCRIPTION

Notifies the application that a connection has reached the state where the TLS sequence has started. This is where the application should exchange handles with the TLS connection. If an AppHandle was previously set to the connection, it will be in the *phAppConn* parameter. This way, the application can keep track if the connection was created by the application

SYNTAX

```
typedef void (RVCALLCONV *
RvSipServerTransportConnectionTlsSequenceStartedEv)(
    IN    RvSipTransportConnectionHandle        hConn,
    INOUT RvSipTransportConnectionAppHandle*    phAppConn);
```

PARAMETERS

hConn

A connection that started the TLS sequence.

phAppConn

The handle given by the application.

RETURN VALUES

None.

RvSipServerTransportConnectionTlsStateChangedEv()

DESCRIPTION

Notifies the application of TLS connection state changes. This callback is called only for TLS state changes and not for connection state changes.

SYNTAX

```
typedef RvStatus (RVCALLCONV *
RvSipServerTransportConnectionTlsStateChangedEv)(
    IN RvSipTransportConnectionHandle        hConnection,
    IN RvSipTransportConnectionAppHandle     hAppConnection,
    IN RvSipTransportConnectionTlsState      eState,
    IN RvSipTransportConnectionStateChangedReason
                                             eReason);
```

PARAMETERS

hConnection

The handle of the connection that changed the TLS state.

hAppHandle

An application handle for the connection.

eState

The connection TLS state.

eReason

The reason for the state change.

RETURN VALUES

Returns RvStatus. (The returned status is ignored in the current SIP Stack version.)

RvSipServerTransportConnectionTlsPostConnectionAssertionEv()

DESCRIPTION

Overrides the default post connection assertion of the SIP Stack. Once a connection has completed the handshake, it is necessary to make sure that the certificate presented was issued for the address for which the connection was made. The SIP Stack automatically performs this assertion. If the application would like to override a failed assertion, it can implement this callback. For example, this callback can be used to compare the host name against a predefined list of outgoing proxies.

SYNTAX

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

PARAMETERS

hConnection

The handle of the connection that changed the TLS state.

hAppConnection

The application handle of the connection.

strHostName

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

hMsg

The message if the connection was asserted against a message.

pbAsserted

RV_TRUE if the connection was asserted successfully. RV_FALSE if the assertion failed. In this case, the connection will be terminated automatically.

RvSipServerTransportBadSyntaxMsgEv()

DESCRIPTION

Notifies the application that a new bad-syntax message was received. The application can fix the message only in this callback and not at a later time. The application should use the *eAction* parameter to decide how the SIP Stack should handle this message—discard it, continue with message processing, or send a 400 response (in case of a Request message). For more information, see the RvSipTransportBsAction enumeration. If the application did not implement this callback, the bad-syntax message will be discarded.

SYNTAX

```
typedef RvStatus (RVCALLCONV *
RvSipServerTransportBadSyntaxMsgEv)(
    IN  RvSipTransportMgrHandle       hTransportMgr,
    IN  RvSipAppTransportMgrHandle    hAppTransportMgr,
    IN  RvSipMsgHandle                hMsgReceived,
    OUT RvSipTransportBsAction        *peAction);
```

PARAMETERS

hTransportMgr

A handle to the *TransportMgr*.

hAppTransportMgr

The application handle. You supply this handle when setting the event handles.

hMsgReveived

The received bad-syntax message.

peAction

The decision of the user on how the SIP Stack should handle this message.

RETURN VALUES

Returns RvStatus. (The returned status is ignored in the current SIP Stack version.)

RvSipServerTransportBadSyntaxStartLineMsgEv()

DESCRIPTION

Notifies the application that a new message was received with a bad-syntax start-line. The application can fix the message only in this callback and not at a later time. The application should use the *eAction* parameter to decide how the SIP Stack should handle this message—discard it, continue with message processing, or send a 400 response (in case of a Request message). For more information, see the RvSipTransportBsAction enumeration. If the application did not implement this callback, the bad-syntax message will be discarded.

SYNTAX

```
typedef RvStatus (RVCALLCONV *
RvSipServerTransportBadSyntaxStartLineMsgEv)(
    IN  RvSipTransportMgrHandle       hTransportMgr,
    IN  RvSipAppTransportMgrHandle    hAppTransportMgr,
    IN  RvSipMsgHandle                hMsgReceived,
    OUT RvSipTransportBsAction        *peAction);
```

PARAMETERS

hTransportMg

A handle to the *TransportMgr*.

hAppTransportMgr

The application handle. You supply this handle when setting the event handles.

hMsgReveived

The received message with the bad-syntax start-line.

peAction

The decision of the user on how the SIP Stack should handle this message.

RETURN VALUES

Returns RvStatus. (The returned status is ignored in the current SIP Stack version.)

RvSipServerTransportConnectionCreatedEv()

DESCRIPTION

Notifies the application about an incoming TCP connection. The callback is called immediately after the connection is accepted. The application can order the SIP Stack to close the connection by means of the *pbDrop* parameter. In this case, the connection will be closed immediately after returning from the callback and its resources will be freed. No data will be received or sent on the connection. If application did not register to this callback, the connection will not be closed, and will be used for data sending and reception.

Note It is not permitted to call the Terminate() API function from this callback.

SYNTAX

```
typedef void (RVCALLCONV *
RvSipServerTransportConnectionCreatedEv)(
   IN  RvSipTransportMgrHandle             hTransportMgr,
   IN  RvSipAppTransportMgrHandle          hAppTransportMgr,
   IN  RvSipTransportConnectionHandle      hConn,
   OUT RvSipTransportConnectionAppHandle   *phAppConn,
   OUT RvBool                              *pbDrop);
```

PARAMETERS

hTransportMgr

A handle to the *TransportMgr*.

hAppTransportMgr

The application handle. You supply this handle when setting the event handles.

hConn

The handle to the created connection.

phAppConn

The handle that the application set for the connection.

pbDrop

If set to RV_TRUE by application, the connection will be dropped immediately after returning from the callback. Otherwise, the connection will not be dropped and will be used for data reception and sending. The default value is RV_FALSE.

Return Values

None.

RvSipServerTransportBufferReceivedEv()

DESCRIPTION

Exposes the raw data buffer to an application that contains exactly one SIP message that was received on the TCP/UDP layer. The application can dump the data by means of this callback. Also, the application can order the SIP Stack to discard the buffer, and not to parse it, by means of the *pbDiscardBuffer* parameter.

SYNTAX

```
typedef void (RVCALLCONV *
RvSipServerTransportBufferReceivedEv)(
  IN  RvSipTransportMgrHandle           hTransportMgr,
  IN  RvSipAppTransportMgrHandle        hAppTransportMgr,
  IN  RvSipTransportLocalAddrHandle     hLocalAddr,
  IN  RvSipTransportAddr                *pSenderAddrDetails,
  IN  RvSipTransportConnectionHandle    hConn,
  IN  RvSipTransportConnectionAppHandle hAppConn,
  IN  RvChar                            *buffer,
  IN  RvUint32                          buffLen,
  OUT RvBool                            *pbDiscardBuffer);
```

PARAMETERS

hTransportMgr

A handle to the *TransportMgr*.

hAppTransportMgr

The application handle. You supply this handle when setting the event handles.

hLocalAddr

The handle to the local address object. This is the address on which the buffer was received.

pSenderAddrDetails

A pointer to RvSipTransportAddr, which contains details of the address from which the message was sent.

hConn

The handle of the connection on which the buffer was received. NULL for UDP.

hAppConn

The handle that the application set for the connection.

buffer

A pointer to the buffer that contains the message.

buffLen

The length of the message in the buffer, in bytes.

bDiscardBuffer

If set to RV_TRUE, the buffer will not be processed and the resources will be freed.

Return Values

None.

RvSipServerTransportBufferToSendEv()

DESCRIPTION

Exposes the raw data buffer to an application, which contains exactly one SIP message, that is going to be sent on TCP/UDP layer. The application can dump the data by means of this callback. Also, the application can decide whether the Transport layer should not transmit the message to its destination.

Note Discarding does not affect the state of the sender object. Discarding a message can simulate message loss on the net.

SYNTAX

```
typedef void (RVCALLCONV *
RvSipServerTransportBufferToSendEv)(
  IN  RvSipTransportMgrHandle             hTransportMgr,
  IN  RvSipAppTransportMgrHandle          hAppTransportMgr,
  IN  RvSipTransportLocalAddrHandle       hLocalAddr,
  IN  RvSipTransportAddr                  *pDestAddrDetails,
  IN  RvSipTransportConnectionHandle      hConn,
  IN  RvSipTransportConnectionAppHandle   hAppConn,
  IN  RvChar                              *buffer,
  IN  RvUint32                            buffLen,
  OUT RvBool                              *pbDiscardBuffer);
```

PARAMETERS

hTransportMgr

A handle to the *TransportMgr*.

hAppTransportMgr

The application handle. You supply this handle when setting the event handles.

hLocalAddr

The handle to the local address object. This is the address from which the buffer is going to be sent.

pDestAddrDetails

A pointer to the RvSipTransportAddr structure, which contains details of the address, to which the message is going to be sent.

hConn

The handle of the connection, on which the buffer is going to be sent. NULL for UDP.

hAppConn

The handle that the application sets for the connection.

buffer

A pointer to the buffer that contains the message.

buffLen

The length of the message in the buffer, in bytes.

bDiscardBuffer

If set to RV_TRUE, the buffer will be not sent and the resources will be freed.

Return Values

None.

RvSipServerTransportConnectionParserResultEv()

DESCRIPTION

The parsing result for the message to an application which arrived over the TCP connection. If the parser encounters bad syntax, the *bLegalSyntax* is RV_FALSE. Otherwise it is RV_TRUE.

SYNTAX

```
typedef void (RVCALLCONV *
RvSipServerTransportConnectionParserResultEv)(
   IN RvSipTransportMgrHandle              hTransportMgr,
   IN RvSipAppTransportMgrHandle           hAppTransportMgr,
   IN RvSipMsgHandle                       hMsg,
   IN RvSipTransportConnectionHandle       hConn,
   IN RvSipTransportConnectionAppHandle    hAppConn,
   IN RvBool                               bLegalSyntax);
```

PARAMETERS

hTransportMgr

A handle to the *TransportMgr*.

hAppTransportMgr

The application handle. You supply this handle when setting the event handles.

hMsg

The handle to the message that was parsed.

hConn

A handle to the connection, on which the message to be parsed, arrived.

hAppConn

The handle that the application sets for the connection.

LegalSyntax

RV_TRUE if the parser did not discover bad syntax. Otherwise, RV_FALSE.

RETURN VALUES

None.

RvSipServerTransportConnectionDataReceivedEv()

DESCRIPTION

Notifies the application that data was read from the socket of the incoming connection.

SYNTAX

```
typedef void (RVCALLCONV *
RvSipServerTransportConnectionDataReceivedEv)(
    IN RvSipServerTransportMgrHandle       hTransportMgr,
    IN RvSipServerTransportMgrAppHandle    hAppTransportMgr,
    IN RvSipTransportConnectionHandle      hConn,
    IN RvSipTransportConnectionAppHandle   hAppConn,
    IN RvChar                              *buff,
    IN RvInt                               buffSize,
    OUT RvBool                             *pbDiscard);
```

PARAMETERS

hTransportMgr

The handle to the TransportMgr.

hAppTransportMgr

The application handle. The application can supply this handle using RvSipServerTransportMgrAttachAppMgr().

hMsg

The handle to the message that was parsed.

hConn

The handle to the connection on which the message was parsed.

hAppConn

The handle that the application set into the connection.

buff

A buffer containing read data.

buffSize

The size of the read data, in bytes.

pbDiscard

If the application sets the parameter to RV_TRUE, the received data will be discarded. If the application does not set any value, the received data will not be discarded.

Return Values

None.

RvSipServerTransportResolveAddressEv()

DESCRIPTION

This callback is fired before any address resolution. The callback allows the application to perform resolution by itself. The application can modify the strIP member of pAddrToResolve to either IP or FQDN (the size is limited to RVSIP_TRANSPORT_LEN_STRING_IP). When pbAddressResolved is set to RV_TRUE, the SIP Server will use pAddrToResolve to send the request. When pbAddressResolved is set to RV_FALSE, or if strIP was set to FQDN, the SIP Server will execute a DNS query for FQDN resolution.

Note The callback is synchronous, meaning the result must be returned in pAddrToResolve within the callback. The returned IP/FQDN MUST not exceed RVSIP_TRANSPORT_LEN_STRING_IP.

SYNTAX

```
typedef RvStatus (RVCALLCONV *
RvSipServerTransportResolveAddressEv)(
  IN    RvSipServerTransportMgrHandle    hTransportMgr,
  IN    RvSipServerTransportMgrAppHandle hAppTransportMgr,
  IN    RvChar*                          szHostName,
  INOUT RvSipTransportAddr*              pAddrToResolve,
  OUT   RvBool*                          pbAddressResolved);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hAppTransportMgr

The application attached handle for *hTransportMgr*.

szHostName

The host name string that should be resolved. (The length of the string is no bigger than RVSIP_MAX_HOST_NAME_LEN).

pAddrToResolve

A structure containing all the information of the address that should be resolved.

pAddrToResolve

The same structure containing all the information of the resolved address.

pbAddressResolved

Indicates whether or not the *pResolvedAddress* was modified by the application.

RETURN VALUES

None.

RvSipServerTransportConnectionServerReuseEv()

DESCRIPTION

When an incoming request has an alias parameter in its top-most Via header, this callback function is called. This callback informs application that a server connection should be reused, and it has to be authorized first. In this callback, the application should authenticate the connection, and if the connection was authorized, the application should call RvSipTransportConnectionEnableConnByAlias(). Calling this function enables the connection for reuse.

SYNTAX

```
typedef RvStatus
(*RvSipServerTransportConnectionServerReuseEv)(
  IN RvSipServerTransportMgrHandle         hTransportMgr,
  IN RvSipServerAppTransportMgrHandle      hAppTransportMgr,
  IN RvSipServerTransportConnectionHandle  hConn,
  IN RvSipServerTransportConnectionAppHandle
                                           hAppConns);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hAppTransportMgr

The application handle. You supply this handle when setting the event handles.

hConn

The handle to the connection that needs to be authorized.

hAppConn

The handle that the application sets for the connection.

RETURN VALUES

None.

RvSipServerTransportRegExpResolutionNeededEv()

DESCRIPTION

When the SIP Server encounters a regular expression as an ENUM resolution result, it uses this callback to ask the application to parse the given unresolved address according to a regular expression retrieved from the DNS server.

SYNTAX

```
typedef RvStatus (RVCALLCONV
*RvSipServerTransportRegExpResolutionNeededEv)(
    IN RvSipServerTransportMgrHandle       hTransportMgr,
    IN RvSipServerTransportMgrAppHandle    hAppTransportMgr,
    IN RvSipTransmitterHandle              hTrx,
    IN RvSipAppTransmitterHandle           hAppTrx,
    INOUT RvSipTransmitterRegExpResolutionParams*
                                           pRegExpParams);
```

PARAMETERS

hTransportMgr

The handle to the *TransportMgr*.

hAppTransportMgr

The application attached handle for *hTransportMgr*.

hTrx

The handle to the transmitter (NULL if the transmitter is not an application transmitter).

hAppTrx

The handle to the transmitter application handle (NULL if the transmitter is not an application transmitter).

pRegExpParams

A structure that holds the information for the regular expression application. RvSipTransmitterRegExpResolutionNeededEv() should fill in the pMatchArray with substring match addresses. Any unused structure elements should contain the -1 value. Each startOffSet element that is not -1 indicates the start offset of the next largest substring match within the string. The relative endOffSet element indicates the end offset of the match.

17

SIP SERVER STD TYPE DEFINITIONS

WHAT'S IN THIS SECTION

This section includes the following SIP Server STD type definitions defined in the *RvSipServerStdList.h*, *RvXMLTypes.h* and *RvSipServerStdBase.h* header files.

The type definitions in this section are:

- STD Handles
- STD Structures and Enumerations
- XML Handles
- XML Structures and Enumerations

STD HANDLES

The handles are:

- RvSipServerStdListHandle
- RvSipServerStdListElemHandle
- RvSipServerStdListElemKeyHandle
- RvSipServerStdListElemDataHandle
- RvSipServerStdAllocatorHandle

RvSipServerStdListHandle

DESCRIPTION

The declaration of a *ListElem* handle.

SYNTAX

```
RV_DECLARE_HANDLE (RvSipServerStdListHandle);
```

RvSipServerStdListElemHandle

DESCRIPTION

The declaration of a *ListElem* handle.

SYNTAX

```
RV_DECLARE_HANDLE (RvSipServerStdListElemHandle);
```

RvSipServerStdListElemKeyHandle

DESCRIPTION

The declaration of a *ListElem* key handle.

SYNTAX

```
RV_DECLARE_HANDLE (RvSipServerStdListElemKeyHandle);
```

RvSipServerStdListElemDataHandle

DESCRIPTION

The declaration of a *ListElem* data handle.

SYNTAX

```
RV_DECLARE_HANDLE (RvSipServerStdListElemDataHandle);
```

RvSipServerStdAllocatorHandle

DESCRIPTION

The Allocator handle.

SYNTAX

```
RV_DECLARE_HANDLE (RvSipServerStdAllocatorHandle);
```

STD STRUCTURES AND ENUMERATIONS

The structures and enumerations are:

- RvSipServerStdListElemIntf
- RvSipServerStdAllocatorIntf
- RvSipServerStdListCfg
- RvXmlResource
- RvXMLMgrCfg
- RvXMLResources

RvSipServerStdListElemIntf

DESCRIPTION

Defines the interface of a *ListElem*.

SYNTAX

```
typedef struct _RvSipServerStdListElemIntf{
    RvUint16                                sizeOfStruct;
    RvSipServerStdAllocatorIntf             allocIntf;
    RvSipServerStdListElemKeyCompareFunc    pfnCompare;
}RvSipServerStdListElemIntf;
```

PARAMETERS

sizeOfStruct

This parameter should be set to the size of RvSipServerStdListElemIntf.

allocIntf

The *Allocator* interface.

pfnCompare

The address of a compare method.

RvSipServerStdAllocatorIntf

DESCRIPTION

Interface functions of an *Allocator*. This interface is used by modules to allocate memory from a client pool. The allocator and free functions will be called with the *Allocator* handle when the module requires to allocate and free memory spaces respectively.

SYNTAX

```
typedef struct _RvSipServerStdAllocatorIntf{
    RvSipServerStdAllocatorHandle       hAllocator;
    RvSipServerStdAllocatorAllocFunc    pfnAlloc;
    RvSipServerStdAllocatorFreeFunc     pfnFree;
}RvSipServerStdAllocatorIntf;
```

PARAMETERS

hAllocator

The handle of the *Allocator*.

pfnAlloc

The address of the allocation function of the *Allocator*.

pfnFree

The address of the free function of the *Allocator*.

RvSipServerStdListCfg

DESCRIPTION

This structure contains the configuration data for a list.

SYNTAX

```
typedef struct
{
    RvInt32                 sizeOfStruct;
    RvSipServerLogHandle    hLog;
    RvBool                  bThreadSafe;
}RvSipServerStdListCfg;
```

PARAMETERS

sizeOfStruct

Size of RvSipServerStdListCfg, in bytes.

hLog

List log instance.

bThreadSafe

RV_TRUE indicates that the list should be thread safe. Otherwise, RV_FALSE.

RvXmlResource

DESCRIPTION

A general resource struct used by the XML encoder module.

SYNTAX

```
typedef struct
{
    RvUint32    numOfAllocatedElements;
    RvUint32    currNumOfUsedElements;
    RvUint32    maxUsageOfElements;
} RvXMLElementResource;
```

RvXMLMgrCfg

SYNTAX

```
typedef struct
{
    /* Pools configuration */
    RvInt32        numXMLDocs;
    RvInt32        numXMLTags;
    RvInt32        numXMLAttrs;
    RvInt32        numPages;
    RvInt32        pageSize;
    /* Number of buffers to allocate for XPath processing */
    RvInt32        XPathNumBuffers;
    /* size of each such buffer */
    RvInt32        XPathBufferSize;
    void*          hApplicationContext;
    /* Logs */
    RvLogMgr*      hLog;
    RvLogSource*   hXMLLog;
}RvXMLMgrCfg;
```

RvXMLResources

DESCRIPTION

Declaration of XML resource structure. This object is used to receive report about the resource consumption of the XML module.

SYNTAX

```
typedef struct
{
RvXMLElementResource XMLDocs;
RvXMLElementResource XMLTags;
RvXMLElementResource XMLAttrs;
RvXMLElementResource XMLPages;
RvXMLElementResource XPathBuffers;
}RvXMLResources;
```

XML HANDLES

The handles are:

- RvXMLMgrHandle
- RvXMLDocHandle
- RvXMLTagHandle
- RvXMLAttrHandle
- RvXMLXPathHandle

RvXMLMgrHandle

DESCRIPTION

The declaration of a handle to an *XMLMgr*. The *XMLMgr* manages all the *XMLDoc*, *XMLTag*, and *XMLAttr* objects.

SYNTAX

```
DECLARE_VOID_POINTER(RvXMLMgrHandle);
```

RvXMLDocHandle

DESCRIPTION

The declaration of a handle to an *XMLDoc*. The *XMLDoc* represents an *XMLDoc* as a tree of *XMLTag* objects and attributes.

SYNTAX

```
DECLARE_VOID_POINTER(RvXMLDocHandle);
```

RvXMLTagHandle

DESCRIPTION

The declaration of a handle to an *XMLTag*. The *XMLTag* represents a tag in the *XMLDoc*. The tag is the basic data structure held by the document as a leaf in a tree.

SYNTAX

```
DECLARE_VOID_POINTER(RvXMLTagHandle);
```

RvXMLAttrHandle

DESCRIPTION

The declaration of a handle to an *XMLAttr*. The *XMLAttr* represents an attribute of a specific tag.

SYNTAX

```
DECLARE_VOID_POINTER(RvXMLAttrHandle);
```

RvXMLXPathHandle

DESCRIPTION

The declaration of a handle to an *XPath* processing job. The handle points to the internal structure (XMLXPathObject) that contains environment/global variables, parsing/evaluating results, and so on.

SYNTAX

```
RV_DECLARE_HANDLE(RvXMLXPathHandle);
```

XML Structures and Enumerations

The structures and enumerations are:

- RvXmlNodeType
- RvXMLListLocation
- RvXMLComparisonFlags
- RvXmlResource
- RvXMLResources

RvXmlNodeType

DESCRIPTION

Represents the type of node (tag).

SYNTAX

```
typedef enum{
    RVXML_ROOT_NODE = 0,
    RVXML_ELEMENT_NODE,
    RVXML_TEXT_NODE,
    RVXML_ATTRIBUTE_NODE,
    RVXML_NAMESPACE_NODE,
    RVXML_PROCINSR_NODE,
    RVXML_COMMENT_NODE,
}RvXMLNodeType;
```

PARAMETERS

RVXML_ROOT_NODE

The root node.

RVXML_ELEMENT_NODE

The element node.

RVXML_TEXT_NODE

The text node.

RVXML_ATTRIBUTE_NODE

The attribute node.

RVXML_NAMESPACE_NODE

The namespace node.

RVXML_PROCINSR_NODE

The processing instruction node.

RVXML_COMMENT_NODE

The comment node.

RvXMLListLocation

DESCRIPTION

Represents the location of an element on a list.

SYNTAX

```
typedef enum{
    RVXML_FIRST_ELEMENT = 0,
    RVXML_LAST_ELEMENT,
    RVXML_NEXT_ELEMENT,
    RVXML_PREV_ELEMENT
}RvXMLListLocation;
```

RvXMLComparisonFlags

DESCRIPTION

Represents the comparison flags available with the XML encoder. The values can be bitwise ORed.

SYNTAX

```
typedef enum {
    RVXML_COMPARE_CASESENSITIVE = 0x00,
    RVXML_COMPARE_CASEINSENSITIVE = 0x01,
    RVXML_COMPARE_TAGATTRIBUTES = 0x02,
    RVXML_COMPARE_TAGRECURSIVE = 0x04,
    RVXML_COMPARE_TAG_NAMESPACE = 0x08,
    RVXML_COMPARE_ATTR_NAMESPACE = 0x10
}RvXMLComparisonFlags;
```

PARAMETERS

RVXML_COMPARE_CASESENSITIVE

The *XMLAttr* or *XMLTag* comparison will be case-sensitive.

RVXML_COMPARE_CASEINSENSITIVE

The *XMLAttr* or *XMLTag* comparison will be case insensitive.

RVXML_COMPARE_TAGATTRIBUTES

The *XMLTag* comparison will include *XMLAttrs*.

RVXML_COMPARE_TAGRECURSIVE

The *XMLTag* comparison will include child *XMLTags*.

RVXML_COMPARE_TAG_NAMESPACE

The comparison will include the namespace of the *XMLTags*.

RVXML_COMPARE_ATTR_NAMESPACE

The comparison will include the namespace of the *XMLAttrs*.

RvXmlResource

DESCRIPTION

A general resource structure used by the XML Module.

SYNTAX

```
typedef struct{
    RvUint32    numOfAllocatedElements;
    RvUint32    currNumOfUsedElements;
    RvUint32    maxUsageOfElements;
}RvXMLElementResource;
```

RvXMLResources

DESCRIPTION

The declaration of the XML resource structure. This object is used to receive reports about the resource consumption of the XML module.

SYNTAX

```
typedef struct{
    RvXMLElementResource    XMLDocs;
    RvXMLElementResource    XMLTags;
    RvXMLElementResource    XMLAttrs;
    RvXMLElementResource    XMLPages;
    RvXMLElementResource    XPathBuffers;
}RvXMLResources;
```

RvXMLXPathValue

DESCRIPTION

Represents the type of the result of an *XPath* expression.

SYNTAX

```
typedef enum _RvXMLXPathValue{
    RVXML_XPATH_VALUE_BOOLEAN = 1,
    RVXML_XPATH_VALUE_INTEGER = 2,
    RVXML_XPATH_VALUE_NUMBER = 3,
    RVXML_XPATH_VALUE_STRING = 4,
    RVXML_XPATH_VALUE_NODE = 5,
    RVXML_XPATH_VALUE_OBJECT
}RvXMLXPathValue;
```

PARAMETERS

RVXML_XPATH_VALUE_BOOLEAN

The Boolean (TRUE/FALSE) is stored in _bool.

RVXML_XPATH_VALUE_INTEGER

The integer is stored in _int.

RVXML_XPATH_VALUE_NUMBER

The float is stored in _flt.

RVXML_XPATH_VALUE_STRING

The pointer to a NULL-terminated string is stored in _sz.

RVXML_XPATH_VALUE_NODE

The node. The handle is stored in _tag.

RVXML_XPATH_VALUE_OBJECT

An unknown type in the case of exceptions.

RvXMLXPathValueSpecial

DESCRIPTION

Represents the type of result special values that are allowed in the *XPath* result.

SYNTAX

```
typedef enum _RvXMLXPathValueSpecial{
    RVXML_XPATH_VALUE_NORMAL = 0,
    RVXML_XPATH_VALUE_NAN,
    RVXML_XPATH_VALUE_INFINITY,
    RVXML_XPATH_VALUE_NEG_INFINITY,
    RVXML_XPATH_VALUE_NEG_ZERO,
}RvXMLXPathValueSpecial;
```

PARAMETERS

RVXML_XPATH_VALUE_NORMAL

A legal value. No additional information.

RVXML_XPATH_VALUE_NAN

The “Not-a-Number” (NaN) value.

RVXML_XPATH_VALUE_INFINITY

Positive infinity.

RVXML_XPATH_VALUE_NEG_INFINITY

Negative infinity.

RVXML_XPATH_VALUE_NEG_ZERO

Negative zero.

XMLXPathNode

DESCRIPTION

Represents the current context of the node enumeration while processing location steps during *XPath* evaluation.

SYNTAX

```
typedef struct _XMLXPathNode{
    RvXMLNodeType      type;
    RvXMLTagHandle     tagBase;
    RvXMLTagHandle     tagCur;
    RvXMLAttrHandle    attrCur;
}XMLXPathNode,*XMLXPathNodePtr;
```

PARAMETERS

RvXMLNodeType

The type of the node.

RvXMLTagHandle

The context node.

RvXMLTagHandle

The current node element.

RvXMLAttrHandle

The current node attribute.

RvXMLXPathVariant

A special data type that can contain any kind of data that may be a result of an evaluated *XPath*. Member RvXMLXPathVariant::size contains:

- The array size, in case of a node set value
- The number of characters, in case of a string value
- Zero, in case of a Boolean value
- Zero (normal number) or a special value index, in case of a number value

(See RvXMLXPathValueSpecial for the special values.)

SYNTAX

```
typedef struct _RvXMLXPathVariant
{
    RvXMLXPathValue    type;
    RvInt16            size;
    RvInt16            typeExt;
    XMLXPathNode       node;
    union              RvXMLXPathVariantValue
    {
        RvInt32             _int;
        RvInt               _bool;
        RvXMLXPathFloat     _flt;
        RvChar*             _sz;
    }var;
}
RvXMLXPathVariant, *RvXMLXPathVariantPtr;
```

PARAMETERS

type

The type of the value. See RvXMLXPathValue.

size

The array/string length or special value index.

typeExt

Extendable information. See RvXMLXPathValueSpecial.

node

A member of the XMLXPathNode type in the RvXMLXPathVariant structure.

RvXMLXPathVariantValue

RvXMLXPathVariant is the result of XPath evaluation. It can be one of the following, according to the RvXMLXPathValue parameter:

- RvXMLXPathVariant is used in case the result is a simple value, such a number or a string.
- XMLXPathNode is used in case the result is an element of the XML, such as an *XMLTag* or an *XMLAttr*.

18

SERVER COMPONENTS TYPE DEFINITIONS

WHAT'S IN THIS SECTION

This section includes the following Server Components type definitions defined in the *RvSipServerSecurityImp.h* and *RvProxyLocationDBImp.h* header files.

The type definitions in this section are:

- Security Server Component Type Definitions
- LocationDB Server Component Type Definitions

Security Server Component Type Definitions

The type definitions are:

- RvSipServerSecurityCfg
- RvSipServerSecurityMgrResource

RvSipServerSecurityCfg

DESCRIPTION

The configuration structure of the Security Server Component.

SYNTAX

```
typedef struct{
    RvInt32                     maxUsers;
    RvChar*                     szPrivateKey;
    RvChar*                     szRealm;
    RvChar*                     szOpaque;
    RvChar*                     szQop;
    RvChar*                     szDomain;
    RvUint8                     logLevel;
    RvChar*                     clientAuthUsername;
    RvChar*                     clientAuthPassword;
    RvChar*                     szNonce;
    RvSipServerDbService        eDbServiceType;
    RvUint32                    maxQueries;
#ifdef RV_SIPSERVER_LDAP
    RvSipServerLdapMgrHandle    hLdapMgr;
    RvChar*                     szPwdAttribName
#endif
}RvSipServerSecurityCfg;
```

PARAMETERS

maxUsers

Defines the maximum number of users in the Security Manager database. If *maxUsers* = –1, the default of RVPROXY_SECURITY_IMP_DEFULT_MAX_USERS is used.

szPrivateKey

Defines the Security Manager private key which is used for nonce generation. If *szPrivateKey* is NULL, the Security Manager handle is used instead as a default.

szRealm

Defines the Security Manager realm value (include <"> <">). If *szRealm* is NULL, the RvSipServerSecurityMgrImpConstruct() function will fail. There is no default parameter available for the realm value.

szOpaque

Defines the Security Manager opaque value (include <"> <">). If *szOpaque* is NULL, no opaque value will be used.

szQop

Defines the Security Manager Qop value (include <"> <">). If *szQop* is NULL, no Qop value will be used.

szDomain

define the sEcurity Manager domain value (include <"> <">). If *szDomain* is NULL, no domain value will be used.

logLevel

The log filter level for the Security Component module.

clientAuthUsername

The user name for client-side authentication (for example when authenticating NOTIFY requests sent by the Presence Server).

clientAuthPassword

The password to use for client-side authentication (for example when authenticating NOTIFY requests sent by the Presence Server).

szNonce

Defines the Security Manager Nonce value (include <"> <">). If *szNonce* is NULL, the Security Manager will generate a random value.

eDbServiceType

The database type to work with while looking for user password. The default value is RVSIPSERVER_DB_SERVICE_LOCAL.

maxQueries

The maximum amount of authentication queries that the Security Server Component can handle.

hLdapMgr

The LDAP manager, required only if *eDbServiceType* is RVSIPSERVER_DB_SERVICE_REMOTE_LDAP.

szPwdAttribName

The name of password attribute of the remote LDAP database.

REMARKS

If the application implemented the RvSipServerSecurityAuthClientSharedSecret() event of the Security interface, it must supply the user name and password that is used when the event is called.

RvSipServerSecurityMgrResource

DEFINITION

The declaration of the Security Server Component resources structure. This object is used to receive a report about the resource consumption of the Security Server Component module.

SYNTAX

```
typedef struct{
    RvSipServerResource    pages;
    RvSipServerResource    users;
}RvSipServerSecurityMgrResource;
```

LocationDB Server Component Type Definitions

The type definitions are:

- RvProxyLocationDBCfg
- RvProxyLocationDBResources

RvProxyLocationDBCfg

DEFINITION

The configuration structure of the Location Database Server Components.

SYNTAX

```
typedef struct{
    RvInt32     maxRegisteredNum;
    RvInt32     maxContactHeaders;
    RvInt32     numPages;
    RvInt32     pageSize;
    RvInt32     numLookupPages;
    RvInt32     lookupPageSize;
    RvUint8     logFilters;
}RvProxyLocationDBCfg;
```

PARAMETERS

maxRegisteredNum

The maximum number of registered users in the *LocationDB*.

maxContactHeaders

The maximum number of bindings for all users in the *LocationDB*.

numPages

The number of pages to allocate for the *LocationDB* memory pool.

pageSize

The size of the pages in the *LocationDB* memory pool.

numLookupPages

The number of pages to allocate for the *LocationDB* lookup memory pool. These pages will be used for lookup requests in case the general memory pool is out of resources. If *maxContactHeaders* = –1, the default of RVPROXY_LOCATIONDB_DEFAULT_NUM_PAGES is used.

lookupPageSize

The size of the pages in the *LocationDB* lookup memory pool. These pages will be used for lookup requests in case the general memory pool is out of resources. If *maxContactHeaders* = –1, the default of RVPROXY_LOCATIONDB_DEFAULT_MAX_CONTACTS is used.

logFilters

The level of log messages in the Location Database Component.

RvProxyLocationDBResources

DEFINITION

Declaration of the Location Database Server Component resources structure. This object is used to receive reports about the resource consumption of the Location Database Server Component module.

SYNTAX

```
typedef struct{
    RvSipServerResource    pages;
    RvSipServerResource    records;
    RvSipServerResource    bindings;
    RvSipServerResource    hashElements;
    RvSipServerResource    hashKeys;
}RvProxyLocationDBResources;
```

19

RPOOL TYPE DEFINITIONS

WHAT'S IN THIS SECTION

This section includes The SIP Stack RPOOL type definitions defined in the *rpool_API.h* header file.

The type definitions in this section are:

HANDLE TYPE DEFINITIONS

The Handle type definitions are:

- HRPOOL
- HPAGE
- NULL_PAGE

HRPOOL

DESCRIPTION

Defines the handle to the memory pool.

SYNTAX

```
RV_DECLARE_HANDLE (HRPOOL);
```

HPAGE

DESCRIPTION

Defines the handle to a page inside a memory pool.

SYNTAX

```
typedef void* HPAGE;
```

NULL_PAGE

DESCRIPTION

Definition for an invalid page.

SYNTAX

```
#define NULL_PAGE (NULL)
```

RPOOL TYPE DEFINITIONS

The RPOOL type definition is:

- RPOOL_Ptr

RPOOL_Ptr

DESCRIPTION

Structure that defines a specific location inside a memory pool. The location is a combination of the pool handle, the page in that pool and a specific offset inside the page.

SYNTAX

```
typedef struct{
    HRPOOL      hPool;
    HPAGE       hPage;
    RvInt32     offset;
}RPOOL_Ptr;
```

20

MIDDLE LAYER TYPE DEFINITIONS

WHAT'S IN THIS CHAPTER

This section provides headers for the Middle layer of the SIP Stack found in the *RvSipMidTypes.h* header file. The Middle layer allows the application to interact with the OS Abstraction layer (Common Core), build application timers, register on file descriptor event, and so on.

The type definitions included in this section are:

- Type Definitions

TYPE DEFINITIONS

The type definitions are:

- RvSipMidMgrHandle
- RvSipMidTimerHandle
- RvSipMidCfg
- RvSipMidSelectEvent
- RvSipMidSelectEv()
- RvSipMidTimerExpEv()
- RvSipMidResources

RvSipMidMgrHandle

DESCRIPTION

Declares a handle to the *Mid-LayerMgr*.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipMidMgrHandle);
```

RvSipMidTimerHandle

DESCRIPTION

Declares a handle to the application timer object.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipMidTimerHandle);
```

RvSipMidCfg

DESCRIPTION

A structure needed to construct the *Mid-LayerMgr*.

SYNTAX

```
typedef struct{
    RvInt32          maxUserFd;
    RvInt32          maxUserTimers;
    RV_LOG_Handle    hLog;
}RvSipMidCfg;
```

PARAMETERS

maxUserFd

The maximal number of user file descriptors.

maxUserTimers

The maximal number of user timers.

hLog

A handle to the log manager.

RvSipMidSelectEvent

DESCRIPTION

Defines select events that can be registered on an application file descriptor.

SYNTAX

```
typedef enum{
    RVSIP_MID_SELECT_READ = 0x01,
    RVSIP_MID_SELECT_WRITE = 0x02
}RvSipMidSelectEvent;
```

PARAMETERS

RVSIP_MID_SELECT_READ

Notifies of the read events.

RVSIP_MID_SELECT_WRITE

Notifies of the write events.

RvSipMidSelectEv()

DESCRIPTION

A callback function that is returned on events. Notifies the application that a read/write event has occurred.

SYNTAX

```
typedef void (RVCALLCONV *RvSipMidSelectEv) (
    IN RvInt                    fd,
    IN RvSipMidSelectEvent      event,
    IN RvBool                   error,
    IN void*                    ctx);
```

PARAMETERS

fd

The handle/file descriptor on which this event occurred.

event

The event that occurred.

error

RV_TRUE if there was an error in the event.

ctx

A context that the application may provide.

RETURN VALUES

None.

RvSipMidTimerExpEv()

DEFINITIONS

A callback function that is returned on events. Notifies the application that a timer expired.

SYNTAX

```
typedef void (RVCALLCONV *RvSipMidTimerExpEv)(
    IN void*    context);
```

PARAMETERS

context

A context that the application may provide.

RvSipMidResources

The resources that the Mid-layer uses. Supply to RvSipMidGetResources() to obtain the resources status of the Mid-layer.

SYNTAX

```
typedef struct{
    RvSipResource    userFds;
    RvSipResource    userTimers;
}RvSipMidResources;
```

PARAMETERS

userFds

The resources status of the Mid-layer file descriptors.

userTimers

The resources status of the Mid-layer timers.

21

SIP Server Common Type Definitions

What's in this Section

This section includes the following Proxy Common type definitions defined in the *RvSipServerCommonTypes.h* header file.

The type definitions in this section are:

- Handles
- Structures and Enumerations

HANDLES

The handles are:

- RvSipServerListHandle
- RvSipServerListElemHandleDescription
- RvSipServerListPoolHandle

RvSipServerListHandle

DESCRIPTION

A declaration of a SIP Server List object (*SipServerList*) handle. This handle represents a *SipServerList* and is needed in all Proxy List functions.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipServerListHandle);
```

RvSipServerListElemHandleDescription

The declaration of a SIP Server List Element object (*SipServerListElem*) handle. This handle represent a single list element. The handle is used in SIP Server List functions such as Push(), Remove(), GetNext().

SYNTAX

```
RV_DECLARE_HANDLE(RvSipServerListElemHandle);
```

RvSipServerListPoolHandle

DESCRIPTION

The declaration of a SIP Server List Pool object (*SipSeerverListPool*) handle. This handle is needed while constructing new SIP Server list.

SYNTAX

```
RV_DECLARE_HANDLE(RvSipServerListPoolHandle);
```

STRUCTURES AND ENUMERATIONS

The structures and enumerations are:

- RvSipServerQueryStatus
- RvSipServerElemLocation
- RvSipServerListElemType
- RvSipServerMode
- RvSipServerDialogOwner
- RvSipServerRecordRouteMode
- RvSipServerExpireVal
- RvSipServerInfoPerEventPackage
- RvSipServerDbService
- RvSipServerLdapTranscOwnerType
- RvSipServerLdapQuery
- RvSipServerActTriggeringServerLayer
- RvSipServerActTriggeringServerObject
- RvSipServerCurrentStatus
- RvSipServerComponentStatus
- RvSipServerLoggingLevel

RvSipServerQueryStatus

DEFINITION

The status of a query.

SYNTAX

```
typedef enum{
    RVSIPSERVER_QUERY_STATUS_UNDEFINED = -1,
    RVSIPSERVER_QUERY_STATUS_FINISHED,
    RVSIPSERVER_QUERY_STATUS_PENDING
}RvSipServerQueryStatus;
```

PARAMETERS

RVSIPSERVER_QUERY_STATUS_FINISHED

The query is finished.

RVSIPSERVER_QUERY_STATUS_PENDING

The query is pending.

RvSipServerElemLocation

DEFINITION

Defines a location in a list.

SYNTAX

```
typedef enum{
    RVPSIPSERVER_FIRST = 0,
    RVPSIPSERVER_LAST,
    RVPSIPSERVER_NEXT,
    RVPSIPSERVER_PREV
}RvSipServerElemLocation;
```

PARAMETERS

RVSIPSERVER_FIRST

The first element.

RVSIPSERVER_LAST

The last element.

RVSIPSERVER_NEXT

Next to an element. Usually when this enumeration is used, the element preceding should be given.

RVSIPSERVER_PREV

Previous to an element. Usually when this enumeration is used, the following element should be given.

RvSipServerListElemType

DESCRIPTION

Defines the SIP Server List element user data type.

SYNTAX

```
typedef enum{
    RVSIPSERVER_ELEM_TYPE_UNDEFINED = -1,
    RVSIPSERVER_ELEM_TYPE_RECORD_ROUTE,
    RVSIPSERVER_ELEM_TYPE_ROUTE,
    RVSIPSERVER_ELEM_TYPE_ADDRESS,
    RVSIPSERVER_ELEM_TYPE_CONTACT,
    RVSIPSERVER_ELEM_TYPE_MSG,
#ifdef RVSIPSERVER_EVENTS
    RVSIPSERVER_ELEM_TYPE_WINFO_SUBS,
    RVSIPSERVER_ELEM_TYPE_PRES_SUBS,
    RVSIPSERVER_ELEM_TYPE_EVENTS_SUBS,
    RVSIPSERVER_ELEM_TYPE_PUBLISH,
    RVSIPSERVER_ELEM_TYPE_PA_PRESENTITY,
#endif /*RVSIPSERVER_EVENTS*/
#ifdef RVSIPSERVER_B2B
    RVSIPSERVER_ELEM_TYPE_B2BDLG
#endif
}RvSipServerListElemType;
```

PARAMETERS

RVPROXY_ELEM_TYPE_UNDEFINED

The user data type is unknown.

RVPROXY_ELEM_TYPE_RECORD_ROUTE

The user data type is a Record Route header.

RVPROXY_ELEM_TYPE_ROUTE

The user data type is a Route header.

RVPROXY_ELEM_TYPE_ADDRESS

The user data type is an Address object.

RVPROXY_ELEM_TYPE_CONTACT

The user data type is a Contact header.

RVPROXY_ELEM_TYPE_MSG

The user data type is a Message object.

RVSIPSERVER_ELEM_TYPE_WINFO_SUBS

The user data type is a *WinfoSubs* object.

RVSIPSERVER_ELEM_TYPE_PRES_SUBS

The user data type is a *PresSubscribe* object.

RVSIPSERVER_ELEM_TYPE_EVENTS_SUBS

The user data type is a *ServerEventsSubs* object.

RVSIPSERVER_ELEM_TYPE_PUBLISH

The user data type is a *PublishServer* object.

RVSIPSERVER_ELEM_TYPE_PA_PRESENTITY

The user data type is a *PAPresentity* object.

RVSIPSERVER_ELEM_TYPE_B2BDLG

The user data type is a *B2BDlg* object.

RvSipServerMode

DESCRIPTION

The modes in which the SIP Server can work.

SYNTAX

```
typedef enum{
    RVSIPSERVER_MODE_UNDEFINED = -1,
    RVSIPSERVER_MODE_STATELESS,
    RVSIPSERVER_MODE_TRANSAC_STATEFUL,
#ifdef RVSIPSERVER_B2B
    RVSIPSERVER_MODE_DIALOG,
#endif /*RVSIPSERVER_B2B*/
    RVSIPSERVER_MODE_DYNAMIC
}RvSipServerMode;
```

PARAMETERS

RVSIPSERVER_MODE_UNDEFINED

The unknown SIP Server mode.

RVSIPSERVER_MODE_STATELESS

The stateless SIP Server mode.

RVSIPSERVER_MODE_STATE_FULL

The stateful SIP Server mode.

RVSIPSERVER_MODE_DIALOG

Handles incoming requests as dialogs.

RVSIPSERVER_MODE_DYNAMIC

The SIP Server mode is dynamic. New incoming requests can be handled as stateless or stateful core objects, or as dialogs (by B2BTransparent or application), as decided by the application. If this mode is selected, the RvProxyPolicyMgrNewRequestRcvdEv() callback must be implemented.

RvSipServerDialogOwner

DESCRIPTION

When working with the SIP Server mode of RVSIPSERVER_MODE_DIALOG or RVSIPSERVER_MODE_DYNAMIC, the parameters listed below determine which module will handle incoming dialogs.

SYNTAX

```
typedef enum{
    RVSIPSERVER_DIALOG_OWNER_UNDEFINED = -1,
    RVSIPSERVER_DIALOG_OWNER_B2B,
    RVSIPSERVER_DIALOG_OWNER_APP,
    RVSIPSERVER_DIALOG_OWNER_NONE,
    RVSIPSERVER_DIALOG_OWNER_DYNAMIC
}RvSipServerDialogOwner;
```

PARAMETERS

RVSIPSERVER_DIALOG_OWNER_UNDEFINED

The owner is undefined.

RVSIPSERVER_DIALOG_OWNER_B2B

The B2B module will handle the incoming dialogs.

RVSIPSERVER_DIALOG_OWNER_APP

The application will handle the incoming dialogs.

RVSIPSERVER_DIALOG_OWNER_NONE

There is no dialog owner for the Stack call-leg. The server dialog waits for the call-leg termination.

RVSIPSERVER_DIALOG_OWNER_DYNAMIC

Each time a new dialog is created, the application can decide whether it will handle the dialog, or that the B2B transparent module will handle it.

RvSipServerRecordRouteMode

DESCRIPTION

The modes in which a *ProxyCoreObj* object can work.

SYNTAX

```
typedef enum{
    RVSIPSERVER_RECORD_ROUTE_MODE_UNDEFINED = -1,
    RVSIPSERVER_RECORD_ROUTE_MODE_NONE,
    RVSIPSERVER_RECORD_ROUTE_MODE_REQ_ONLY,
    RVSIPSERVER_RECORD_ROUTE_MODE_REQ_AND_RESP_REWRITE
}RvSipServerRecordRouteMode;
```

PARAMETERS

RVSIPSERVER_RECORD_ROUTE_MODE_UNDEFINED

Unknown proxy record route mode.

RVSIPSERVER_RECORD_ROUTE_MODE_NONE

Does not insert a Record Route.

RVSIPSERVER_RECORD_ROUTE_MODE_REQ_ONLY

Adds Record Route headers in requests only.

RVSIPSERVER_RECORD_ROUTE_MODE_REQ_AND_RESP_REWRITE

Adds Record Route headers in requests and rewrites in responses.

RvSipServerExpireVal

DESCRIPTION

The expiration type that should be used for accepting requests.

SYNTAX

```
typedef enum{
    RVSIPSERVER_EXPIRE_VAL_UNDEFINED,
    RVSIPSERVER_EXPIRE_VAL_DEFAULT,
    RVSIPSERVER_EXPIRE_VAL_REQ,
    RVSIPSERVER_EXPIRE_VAL_APP
}RvSipServerExpireVal;
```

PARAMETERS

RVSIPSERVER_EXPIRE_VAL_DEFAULT

The default value from configuration.

RVSIPSERVER_EXPIRE_VAL_REQ

The value from the incoming request.

RVSIPSERVER_EXPIRE_VAL_APP

The value that the application supplies in the API.

RvSipServerInfoPerEventPackage

DESCRIPTION

A list of valid content types per event package and template.

SYNTAX

```
typedef struct{
    RvChar*    eventPackage;
    RvChar*    eventTemplate;
    RvChar*    validContentTypes;
}RvSipServerInfoPerEventPackage;
```

PARAMETERS

eventPackage

The package name in string format.

eventTemplate

The template name.

validContentTypes

The list of valid content types, separated by a semicolon (;).

RvSipServerDbService

DESCRIPTION

Defines database services to use for querying security or location services.

SYNTAX

```
typedef enum{
    RVSIPSERVER_DB_SERVICE_UNDEFINE =    -1,
    RVSIPSERVER_DB_SERVICE_LOCAL,   =     2,
#ifdef RVSIPSERVER_LDAP
    RVSIPSERVER_DB_SERVICE_REMOTE_LDAP  = 3
#endif /*RVSIPSERVER_LDAP*/
}RvSipServerDbService;
```

PARAMETERS

RVSIPSERVER_DB_SERVICE_UNDEFINE

The database service to use was not defined.

RVSIPSERVER_DB_SERVICE_LOCAL

Uses the local database for querying.

RVSIPSERVER_DB_SERVICE_REMOTE_LDAP

Uses the LDAP database for querying.

RvSipServerLdapTranscOwnerType

DESCRIPTION

Defines the owner of the *LDAPTransc*. The owner is the object that caused the creation and the execution of the *LDAPTransc*.

SYNTAX

typedef enum{

```
    RVSIPSERVER_LDAP_TRANSC_OWNER_UNDEFINE = -1,
    RVSIPSERVER_LDAP_TRANSC_OWNER_REG_SERVER,
    RVSIPSERVER_LDAP_TRANSC_OWNER_CORE_OBJ,
    RVSIPSERVER_LDAP_TRANSC_OWNER_PA,
    RVSIPSERVER_LDAP_TRANSC_OWNER_APP
}RvSipServerLdapTranscOwnerType;
```

PARAMETERS

RVSIPSERVER_LDAP_TRANSC_OWNER_UNDEFINE

The *LDAPTransc* owner is not defined.

RVSIPSERVER_LDAP_TRANSC_OWNER_REG_SERVER

The *LDAPTransc* owner is the *RegisterServer* processing an incoming REGISTER request.

RVSIPSERVER_LDAP_TRANSC_OWNER_CORE_OBJ

The *LDAPTransc* owner is a *ProxyCoreObj* executing a look-up query.

RVSIPSERVER_LDAP_TRANSC_OWNER_PA

The *LDAPTransc* owner is a Presence Agent executing a look-up query.

RVSIPSERVER_LDAP_TRANSC_OWNER_APP

The *LDAPTransc* owner is the application.

RvSipServerLdapQuery

DESCRIPTION

Defines the currently supported LDAP queries.

SYNTAX

```
typedef enum{
    RVSIPSERVER_LDAP_QUERY_UNDEFINE = -1,
    RVSIPSERVER_LDAP_QUERY_ADD      =  0,
    RVSIPSERVER_LDAP_QUERY_REMOVE   =  1,
    RVSIPSERVER_LDAP_QUERY_UPDATE   =  2,
    RVSIPSERVER_LDAP_QUERY_SEARCH   =  3
}RvSipServerLdapQuery;
```

PARAMETERS

RVSIPSERVER_LDAP_QUERY_UNDEFINE

The query type is not defined.

RVSIPSERVER_LDAP_QUERY_ADD

The query adds records to the LDAP engine.

RVSIPSERVER_LDAP_QUERY_REMOVE

The query removes records from the LDAP engine.

RVSIPSERVER_LDAP_QUERY_UPDATE

The query updates records of the LDAP engine.

RVSIPSERVER_LDAP_QUERY_SEARCH

The query searches records on the LDAP engine.

RvSipServerActTriggeringServerLayer

DESCRIPTION

Represents the underlay layer that triggers the accounting record generation.

SYNTAX

```
typedef enum{
    RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_UNDEFINE = -1,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_PROXY,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_REGISTRAR,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_EVENT_SERVER,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_PRES,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_WINFO,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_PUBLISH,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_B2B,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_SERVER_DIALOG
}RvSipServerActTriggeringServerLayer;
```

PARAMETERS

RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_UNDEFINE

The underlay layer is not defined.

RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_PROXY

The Proxy core layer.

RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_REGISTRAR

The Registrar layer.

RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_EVENT_SERVER

The Event layer.

RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_PRES

The Presence layer.

RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_WINFO

The Winfo layer.

RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_PUBLISH

The Publish layer.

RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_B2B

The B2B layer.

RVSIPSERVER_ACT_TRIGGERING_SERVER_LAYER_SERVER_DIALOG

The Server Dialog layer.

RvSipServerActTriggeringServerObject

DESCRIPTION

Represents the underlay object type that triggers the accounting record generation.

SYNTAX

```
typedef enum{
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_UNDEFINE = -1,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_PROXY_CORE,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_REG_SERVER,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_EVENTS_SUBS,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_EVENTS_NOTIFY,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_PRES_SUBS,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_PRES_NOTIFY,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_WINFO_SUBS,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_WINFO_NOTIFY,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_PUBLISH,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_B2B,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_B2BDLG,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_B2BTRANSC,
    RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_SERVER_DLG
}RvSipServerActTriggeringServerObject;
```

PARAMETERS

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_UNDEFINE

The underlay layer is not defined.

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_PROXY_CORE

A Proxy Core object (*ProxyCoreObj*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_REG_SERVER

A Register Server object (*RegServer*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_EVENTS_SUBS

An Events Subscribe object (*EventsSubs*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_EVENTS_NOTIFY

An Events Notify object (*EventsNotify*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_PRES_SUBS

A Presence Subscribe object (*PresSubscribe*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_PRES_NOTIFY

A Presence Notify object (*PresNotify*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_WINFO_SUBS

A Winfo Subscribe object (*WinfoSubs*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_WINFO_NOTIFY

A Winfo Notify object (*WinfoNotify*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_PUBLISH

A Publish object (*Publish*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_B2B

A B2B object (*B2B*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_B2BDLG

A B2B Dialog object (*B2BDialog*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_B2BTRANSC

A B2B Transaction object (*B2BTrans*).

RVSIPSERVER_ACT_TRIGGERING_SERVER_OBJ_SERVER_DLG

A Server Dialog object (*ServerDialog*).

RvSipServerCurrentStatus

DESCRIPTION

Represents the current status of the SIP Server.

SYNTAX

```
typedef enum{
    RVSIPSERVER_STATUS_UNDEFINED = -1,
    RVSIPSERVER_STATUS_OK,
    RVSIPSERVER_STATUS_REBOOT,
}RvSipServerCurrentStatus;
```

RvSipServerComponentStatus

DESCRIPTION

Represents the current status of a component of the SIP Server.

SYNTAX

```
typedef enum{
    RVSIPSERVER_COMPONENT_UNDEFINED = -1,
    RVSIPSERVER_COMPONENT_DISABLED,
    RVSIPSERVER_COMPONENT_ON,
    RVSIPSERVER_COMPONENT_OFF,
}RvSipServerComponentStatus;
```

RvSipServerLoggingLevel

DESCRIPTION

Defines the log levels to be used by the SIP Server modules. Selecting one log level automatically selects all of the above levels. For instance, selecting the ERROR level will automatically select the EXCEPTION level as well.

SYNTAX

```
typedef enum{
    RVSIPSERVER_LOGGING_LEVEL_DEFAULT = -1,
    RVSIPSERVER_LOGGING_LEVEL_NONE, /*0*/
    RVSIPSERVER_LOGGING_LEVEL_EXCEPTION, /*1*/
    RVSIPSERVER_LOGGING_LEVEL_ERROR, /*2*/
    RVSIPSERVER_LOGGING_LEVEL_WARNING, /*3*/
    RVSIPSERVER_LOGGING_LEVEL_INFO, /*4*/
    RVSIPSERVER_LOGGING_LEVEL_DEBUG, /*5*/
    RVSIPSERVER_LOGGING_LEVEL_INOUT, /*6*/
}RvSipServerLoggingLevel;
```

22

STATUS CODES

The status codes in Table 22-1 include SIP specific codes defined in the *RV_SIP_DEF.h* header file and the general status codes defined in the *rverror.h* header file. The status codes may be returned by SIP Server API functions as RvStatus.

.

Table 22-1 *Status Codes*

Value	Meaning
RV_OK	The function was completed successfully.
RV_ERROR_UNKNOWN	Failure of unspecified type.
RV_ERROR_OUTOFRESOURCES	The function can not be executed due to limited resources.
RV_ERROR_BADPARAM	A parameter passed to a function is illegal.
RV_ERROR_NULLPTR	The required pointer parameter was a NULL pointer.
RV_ERROR_OUTOFRANGE	A parameter that was passed to a function is out of range.
RV_ERROR_DESTRUCTED	The referred object was already terminated.

Table 22-1 *Status Codes*

Value	Meaning
RV_ERROR_NOTSUPPORTED	The request is not supported under the current configuration.
RV_ERROR_UNINITIALIZED	The object is uninitialized.
RV_ERROR_TRY_AGAIN	The action cannot be completed—try again later.
RV_ERROR_ILLEGAL_ACTION	The requested action is illegal—usually an illegal action occurring in the current state.
RV_ERROR_NETWORK_ PROBLEM	Action failed due to network problems.
RV_ERROR_INVALID_HANDLE	A handle passed to a function is illegal.
RV_ERROR_NOT_FOUND	The requested item cannot be found.
RV_ERROR_INSUFFICIENT_ BUFFER	The buffer is too small.
RV_ERROR_ILLEGAL_SYNTAX	The parser identified a syntax error, or a parser error occurred.
RV_ERROR_OBJECT_ALREADY _EXISTS	An object with this unique set of values already exists.
RV_ERROR_COMPRESSION_ FAI LURE	The compression operation failed.
RV_ERROR_NUM_OF_THREADS _DECREASED	The processing thread exited abnormally.

INDEX

H

N

R

X