Legal Notices

© Copyright 2007 Hewlett-Packard Development Company, L.P.

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation and Technical Data for Commercial Items are licensed to the U.S. Government under vendor’s standard commercial license.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

UNIX is a registered trademark of The Open Group.
## CONTENTS

### STACK MANAGER MODULE

1. Stack Manager Functions
   - Stack Manager Control Functions 4
   - Stack Manager Get and Set Functions 11

### CALL-LEG MODULE

2. Call-leg Functions
   - Call-leg Manager Functions 36
   - Call-leg Session Functions 42
   - Call-leg Get and Set Functions 70
   - Call-leg High Availability Functions 114
   - Call-leg Server Authentication Functions 119
   - Call-leg Replaces Functions 130
   - Call-leg Session-Timer Functions 133
   - Call-leg DNS Functions 150
   - Call-leg Local and Remote Address Functions 159
   - Call-leg Forking Functions 168
TRANSACTION MODULE

3  Transaction Functions
   Transaction Manager Functions 176
   Transaction Control Functions 184
   Transaction Get and Set Functions 199
   Transaction Authentication Functions 242
   Transaction DNS Functions 249
   Transaction Local and Remote Address Functions 253

TRANSMITTER MODULE

4  Transmitter Functions
   Transmitter Manager Functions 266
   Transmitter Control Functions 273
   Transmitter Get and Set Functions 280

REGISTER-CLIENT MODULE

5  Register-Client Functions
   Register-Client Manager Functions 312
   Register-Client Control Functions 318
   Register-Client Get And Set Functions 327
   Register-Client DNS Functions 357
   Register-Client Local and Remote Address Functions 362
   Register-Client Authentication Functions 369
AUTHENTICATOR MODULE

6 Authenticator Functions
Authenticator Control Functions 378
Authenticator Get and Set Functions 390

EVENT NOTIFICATION MODULE

7 Event Notification Functions
Subscription Manager API 404
Subscription Control API 413
Subscription Get and Set API 435
Subscription Notification API 461
Subscription Authentication API 472
Subscription DNS API 481
Subscription Refer API 486
Subscription High Availability API 498
Subscription Forking API 502

TRANSPORT MODULE

8 Transport Functions
Transport Functions 510
DNS Transport Functions 577
RESOLVER MODULE

9 Resolver Functions
   Resolver Manager Functions  604
   Resolver Control Functions  614
   Resolver Get and Set Functions  618

MEMORY POOL MODULE

10 RPOOL Functions
   RPOOL Control Functions  624

MID-LAYER MODULE

11 Mid-layer Functions
   Mid-layer Control Functions  638

COMMON MODULE

12 Common List Functions
   Common List Control Functions  668
   Common List Get and Set Functions  675
TYPE DEFINITIONS AND STATUS CODES

13 Call-leg API Type Definitions
Handle Type Definitions 684
Call-leg Type Definitions 690
Call-leg Callback Functions 717

14 Stack API Type Definitions
Handle Type Definitions 752
Stack Manager Type Definitions 755
Configuration Parameters 762
Stack Manager Callback Functions 768

15 Transaction API Type Definitions
Handle Type Definitions 772
Transaction Type Definitions 777
Transaction Callback Functions 788
Transaction Manager Callback Functions 804

16 Transmitter API Type Definitions
Handle Type Definitions 810
Transmitter Type Definitions 814
Transmitter Callback Functions 823

17 Register-Client API Type Definitions
Handle Type Definitions 834
Register-Client Type Definitions 838
Register-Client Callback Functions 845
<table>
<thead>
<tr>
<th>Chapter</th>
<th>API Type Definitions</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>Authenticator API Type Definitions</td>
<td>856</td>
</tr>
<tr>
<td></td>
<td>Handle Type Definitions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Authenticator Type Definitions</td>
<td>860</td>
</tr>
<tr>
<td></td>
<td>Authenticator Callback Functions</td>
<td>863</td>
</tr>
<tr>
<td>19</td>
<td>Event Notification API Type Definitions</td>
<td>876</td>
</tr>
<tr>
<td></td>
<td>Handle Type Definitions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Subscription Type Definitions</td>
<td>882</td>
</tr>
<tr>
<td></td>
<td>Subscription Callback Functions</td>
<td>902</td>
</tr>
<tr>
<td></td>
<td>Notification Callback Functions</td>
<td>907</td>
</tr>
<tr>
<td></td>
<td>Authentication Callback Functions</td>
<td>913</td>
</tr>
<tr>
<td></td>
<td>Message Callback Functions</td>
<td>918</td>
</tr>
<tr>
<td></td>
<td>REFER Callback Functions</td>
<td>921</td>
</tr>
<tr>
<td></td>
<td>General Callback Functions</td>
<td>924</td>
</tr>
<tr>
<td>20</td>
<td>Transport API Type Definitions</td>
<td>934</td>
</tr>
<tr>
<td></td>
<td>Transport Handle Type Definitions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transport Type Definitions</td>
<td>944</td>
</tr>
<tr>
<td></td>
<td>Transport DNS Type Definitions</td>
<td>974</td>
</tr>
<tr>
<td></td>
<td>Transport Callback Functions</td>
<td>978</td>
</tr>
<tr>
<td></td>
<td>Transport Object Event Handler</td>
<td>1008</td>
</tr>
<tr>
<td>21</td>
<td>Resolver API Type Definitions</td>
<td>1012</td>
</tr>
<tr>
<td></td>
<td>Handle Type Definitions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Resolver Type Definitions</td>
<td>1016</td>
</tr>
<tr>
<td></td>
<td>Resolver Callback Functions</td>
<td>1020</td>
</tr>
<tr>
<td>22</td>
<td>RPOOL Type Definitions</td>
<td>1024</td>
</tr>
<tr>
<td></td>
<td>Handle Type Definitions</td>
<td></td>
</tr>
</tbody>
</table>
About This Document

This document lists the Application Programming Interfaces (API) functions of the SIP Stack. Each function has a general description, syntax description, and other important information. The Reference Guide also contains the different enumerations and structure types that are defined in the API.

The document printing date and part number indicate the document’s current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The document part number will change when extensive changes are made.

The latest version of the document will be available at:

Document updates can be issued between editions to correct errors or document product changes. To ensure that you receive the updated or new edition, subscribe to the appropriate support service.

Contact your HP sales representative for details.

Intended Audience

This document is intended for developers who wish to build SIP Stack applications. Readers are expected to be familiar with the following:

- System administration concepts
- UNIX operating system concepts
- Networking concepts

Document Organization

The HP-UX C SIP Stack Reference Guide is organized as follows:

Chapter 1  Stack Manager Module Describes the Stack Manager API functions included in the RvSipStack.h header file.

Chapter 2  Call-leg Module Describes the Call-leg API functions included in the RvSipCallLeg.h header file.

Chapter 3  Transaction Module Describes the Transaction API functions included in the RvSipTransaction.h header file.

Chapter 4  Transmitter Module Describes the transmitter functions found in the RvSipTransmitter.h header file.
Chapter 5  **Register-Client Module** Describes the Register-Client API functions found in the `RvSipRegClient.h` header file.

Chapter 6  **Authenticator Module** Describes the Authenticator API functions found in the `RvSipAuthenticator.h` header file.

Chapter 7  **Event Notification Module** Describes the Event Notification functions found in the `RvSipSubscription.h` header file.

Chapter 8  **Transport Module** Describes the Transport API functions found in the `RvSipTransport.h` and `RvSipTransportDNS.h` header files.

Chapter 9  **Resolver Module** Describes the Resolver API functions found in the `RvSipResolver.h` header file.

Chapter 10  **Memory Pool Module** Describes the RPOOL functions found in the `rpool_API.h` header file.

Chapter 11  **Mid-layer Module** Describes the Mid-layer API functions found in the `RvSipMid.h` header file.

Chapter 12  **Common Module** Describes an API implementation of the SIP Common List found in the `RvSipCommonList.h` header file.

Chapter 13  **Call-leg API Type Definitions** Describes the SIP Stack Call-leg type definitions and callback functions defined in the `RvSipCallLegTypes.h` header file.

Chapter 14  **Stack API Type Definitions** Describes the SIP Stack type definitions and configuration parameters defined in the `RvSipStackTypes.h` file.

Chapter 15  **Transaction API Type Definitions** Describes SIP Stack Transaction type definitions and callback functions defined in the `RvSipTransactionTypes.h` header file.

Chapter 16  **Transmitter API Type Definitions** Describes the SIP Stack Transmitter type definitions and callback functions defined in the `RvSipTransmitterTypes.h` header file.

Chapter 17  **Register-Client API Type Definitions** Describes the SIP Stack Register-Client type definitions and callback functions defined in the `RvSipRegClientTypes.h` header file.

Chapter 18  **Authenticator API Type Definitions** Describes the SIP Stack Authenticator type definitions and callback functions defined in the `RvSipAuthenticator.h` header file.
Chapter 19  **Event Notification API Type Definitions**  Describes the SIP Stack Event Notification type definitions and callback functions type definitions defined in the *RvSipSubscriptionTypes.h* header file.

Chapter 20  **Transport API Type Definitions**  Describes the SIP Stack Transport type definitions and callback functions defined in the *RvSipTransportTypes.h* and *RvSipTransportDNSTypes.h* header files.

Chapter 21  **Resolver API Type Definitions**  Describes the SIP Stack Resolver type definitions and callback functions defined in the *RvSipResolverTypes.h* header file.

Chapter 22  **RPOOL Type Definitions**  Describes the SIP Stack RPOOL type definitions defined in the *rpool_API.h* header file.

Chapter 23  **Mid-layer Type Definitions**  Describes the headers for the Mid-layer of the SIP Stack found in the *RvSipMidTypes.h* header file.

Chapter 24  **Common Type Definitions**  Describes the Common type definitions that are used by the different SIP Stack layers, which are defined in the *RvSipCommonTypes.h* and *RvSipCommonList.h* header files.

Chapter 25  **Resource Type Definitions**  Describes the SIP Stack Resources type definitions defined in the *RvSipResourcesTypes.h* header file.

Chapter 26  **Status Codes**  Describes the SIP specific codes defined in the *RV_SIP_DEF.h* header file and the general status codes defined in the *rverror.h* header file.

**Typographic Conventions**

This document uses the following typographic conventions:

**monospace**  Computer output, files, directories, software elements such as command options, function names, and parameters. Read tunables from the */etc/vx/tunefstab* file.

**italic**  New terms, book titles, emphasis, and variables replaced with a name or value.

%  C shell prompt

$  Bourne/Korn shell prompt
# Superuser prompt (all shells)
\Continued input on the following line; you do not type this character.
[ ] In command synopsis, brackets indicate an optional argument.
ls [ -a ]
|In command synopsis, a vertical bar separates mutually exclusive arguments.
mount [ suid | nosuid ]
Ctrl+A This symbol indicates that you hold down the first named key while pressing the key or mouse button that follows the plus.

Related Information
Additional information about SIP Stack is available at:
http://docs.hp.com
This website contains the following documents about SIP Stack:
- HP-UX SIP Release Notes
- HP-UX C SIP Stack Reference Guide
- HP-UX C SIP Stack Messages Layer Reference Guide
- HP-UX Java SIP Stack Programmer’s Guide

HP Welcomes Your Comments
HP welcomes your comments concerning this document. We are committed to providing documentation that meets your needs.
Send your comments or suggestions to: feedback@fc.hp.com
Include the document title, manufacturing part number, and any comment on the error found in this document. Also, include what we did right, so we can incorporate it into other documents.
The Stack Manager API provides functions for the Stack Manager module of the SIP Stack. It contains functions that initialize, construct and destruct the SIP Stack.

This part contains the following section:

- Stack Manager Functions
The Stack Manager API functions enable you to initialize, construct and destruct the SIP Stack. This section contains the Stack Manager API functions included in the `RvSipStack.h` header file.

The functions included in this section are:

- Stack Manager Control Functions
- Stack Manager Get and Set Functions
Stack Manager Control Functions

The Control functions are as follows:

- `RvSipStackInitCfg()`
- `RvSipStackConstruct()`
- `RvSipStackProcessEvent()`
- `RvSipStackSelect()`
- `RvSipStackSelectUntil()`
- `RvSipStackDestruct()`
**RvSipStackInitCfg()**

**DESCRIPTION**

Initializes the **RvSipStackCfg** structure. This structure is given to the **RvSipStackConstruct()** function and it is used to initialize the SIP Stack. The **RvSipStackInitCfg()** function relates to two types of parameters found in the **RvSipStackCfg** structure:

- A. Parameters that influence the value of other parameters
- B. Parameters that are influenced by the value of other parameters

The **RvSipStackInitCfg()** sets all parameters of type A to default values and parameters of type B to –1. For example, the **maxCallLegs** is of type A and is set to 10. The **maxTransaction** is of type B and is set to –1. When calling the **RvSipStackConstruct()** function, all the B type parameters will be calculated using the values found in the A type parameters. If you change the A type values, the B type values will be changed accordingly.

**SYNTAX**

```c
void RvSipStackInitCfg(
    IN RvUint8 sizeOfCfg,
    OUT RvSipStackCfg *pStackCfg);
```

**PARAMETERS**

- **sizeOfCfg**
  The size of the configuration structure.

- **pStackCfg**
  The configuration structure containing the SIP Stack default values.

**RETURN VALUES**

None.
RvSipStackConstruct()

DESCRIPTION
Constructs and initializes the SIP Stack. This function allocates the required memory and constructs SIP Stack objects according to configuration parameters specified in the RvSipStackCfg structure. The function returns a handle to the Stack Manager (StackMgr). You need this handle in order to use the Stack Manager API functions.

SYNTAX
RvStatus RvSipStackConstruct(
    IN RvInt32 sizeOfCfg,
    INOUT RvSipStackCfg *pStackCfg,
    OUT RvSipStackHandle *hStack);

PARAMETERS
sizeOfCfg
The size of the configuration structure.

pStackCfg
A structure containing SIP Stack configuration parameters. This parameter is also an output parameter. Values that were set to –1 will be replaced with a value that the SIP Stack calculates. The SIP Stack default values will replace invalid values.

hStack
The handle to the StackMgr.

RETURN VALUES
Returns RvStatus.
RvSipStackProcessEvents()

DESCRIPTION
Checks for events and processes them as they occur.

Note  You must call this function from the same thread that constructed the SIP Stack.

SYNTAX
void RvSipStackProcessEvents();

PARAMETERS
None.

RETURN VALUES
None.
Stack Manager Control Functions
RvSipStackSelect()

**RvSipStackSelect()**

**DESCRIPTION**
Performs one call to the select() function on HP-UX operating system supporting the Select interface. The main loop of the application can be:
```c
while (1){
    RvSipStackSelect();
}
```

**SYNTAX**
```
RvStatus RvSipStackSelect(void);
```

**PARAMETERS**
None.

**RETURN VALUES**
Returns `RvStatus`.

**REMARKS**
You must call this function from the same thread that constructed the SIP Stack.
**RvSipStackSelectUntil()**

**DESCRIPTION**
Performs one call to the select() function on HP-UX operating system supporting the Select interface. This function also gives the application the ability to give a maximum blocking delay.

**SYNTAX**
```
RvStatus RvSipStackSelectUntil(
    IN RvUint32 delay);
```

**PARAMETERS**
- **delay**
The maximum time to block in milliseconds.

**RETURN VALUES**
Returns **RvStatus**.

**REMARKS**
You must call this function from the same thread that constructed the SIP Stack.
**RvSipStackDestruct()**

**DESCRIPTION**
Destructor. The function destructs all SIP Stack objects and frees all allocated memory.

**Note** You must call this function from the same thread that constructed the SIP Stack.

**SYNTAX**
```
RvStatus RvSipStackDestruct(
    IN RvSipStackHandle   hStack);
```

**PARAMETERS**

**hStack**
The handle to the SIP Stack.

**RETURN VALUES**
Returns **RvStatus**.
Stack Manager Get and Set Functions

The Get and Set functions are as follows:

- RvSipStackGetVersion()
- RvSipStackSetAppHandle()
- RvSipStackGetAppHandle()
- RvSipStackGetCallLegMgrHandle()
- RvSipStackGetTransactionMgrHandle()
- RvSipStackGetMsgMgrHandle()
- RvSipStackGetRegClientMgrHandle()
- RvSipStackGetAuthenticatorHandle()
- RvSipStackGetTransportMgrHandle()
- RvSipStackGetSubsMgrHandle()
- RvSipStackGetResolverMgrHandle()
- RvSipStackGetLogHandle()
- RvSipStackSetNewLogFilters()
- RvSipStackIsLogFilterExist()
- RvSipStackGetResources()
- RvSipStackGetStatistics()
- RvSipStackMgrIsEnhancedDnsFeatureEnabled()
- RvSipStackMgrIsTlsFeatureEnabled()
- RvSipStackIsSessionTimerSupported()
- RvSipStackIsReplacesSupported()
Stack Manager Get and Set Functions
RvSipStackGetVersion()

---

RvSipStackGetVersion()

DESCRIPTION
Gets the current version of the SIP Stack.

SYNTAX
RvChar * RvSipStackGetVersion();

PARAMETERS
None.

RETURN VALUES
Returns the version number.
RvSipStackSetAppHandle()

**DESCRIPTION**
Sets an application handle to the SIP Stack instance. Setting a SIP Stack application handle can be useful when implementing multi-instance applications.

**SYNTAX**

```c
RvStatus RvSipStackSetAppHandle(
    IN RvSipStackHandle hStack,
    IN RvSipAppStackHandle hAppStack);
```

**PARAMETERS**

- **hStack**
The handle to the SIP Stack.

- **hAppStack**
A new application handle to the SIP Stack.

**RETURN VALUES**

Returns *RvStatus*.
RvSipStackGetAppHandle()

DESCRIPTION
Returns the application handle to this SIP Stack instance.

SYNTAX
RvStatus RvSipStackGetAppHandle(
    IN  RvSipStackHandle    hStack,
    OUT RvSipAppStackHandle *phAppStack);

PARAMETERS
hStack
The handle to the SIP Stack.

phAppStack
The application handle to the SIP Stack instance.

RETURN VALUES
Returns RvStatus.
RvSipStackGetCallLegMgrHandle()

DESCRIPTION

Gets the Call-legMgr handle. You need this handle in order to use the Call-leg API functions.

SYNTAX

RvStatus RvSipStackGetCallLegMgrHandle(
    IN RvSipStackHandle hStack,
    OUT RvSipCallLegMgrHandle *phCallLegMgr);

PARAMETERS

hStack

The handle to the SIP Stack.

phCallLegMgr

The handle to the Call-legMgr.

RETURN VALUES

Returns RvStatus.
Stack Manager Get and Set Functions
RvSipStackGetTransactionMgrHandle()

RvSipStackGetTransactionMgrHandle()

DESCRIPTION
Gets the TransactionMgr handle. You need this handle in order to use the Transaction API functions.

SYNTAX
RvStatus RvSipStackGetTransactionMgrHandle(  
    IN RvSipStackHandle hStack,
    OUT RvSipTranscMgrHandle *phTranscMgr);

PARAMETERS
hStack
The handle to the SIP Stack.

phTranscMgr
The handle to the TransactionMgr.

RETURN VALUES
Returns RvStatus.
RvSipStackGetMsgMgrHandle()

DESCRIPTION

Gets the MessageMgr handle. You need this handle in order to use the Message API functions.

SYNTAX

RvStatus RvSipStackGetMsgMgrHandle(
    IN RvSipStackHandle hStack,
    OUT RvSipMsgMgrHandle *phMsgMgr);

PARAMETERS

hStack
The handle to the SIP Stack.

phMsgMgr
The handle to the MessageMgr.

RETURN VALUES

Returns RvStatus.
**Stack Manager Get and Set Functions**

**RvSipStackGetRegClientMgrHandle()**

**DESCRIPTION**

Gets the *Register-client Mgr* handle. You need this handle in order to use the Register-Client API functions.

**SYNTAX**

```c
RvStatus RvSipStackGetRegClientMgrHandle(
    IN RvSipStackHandle hStack,
    OUT RvSipRegClientMgrHandle *phRegClientMgr);
```

**PARAMETERS**

- **hStack**
  The handle to the SIP Stack.

- **phRegClientMgr**
  The handle to the *Register-client Mgr*.

**RETURN VALUES**

Returns *RvStatus*.
Stack Manager Get and Set Functions
RvSipStackGetAuthenticatorHandle()

**RvSipStackGetAuthenticatorHandle()**

**DESCRIPTION**

Gets the *authenticator* handle. You need this handle in order to use the Authenticator API.

**SYNTAX**

```
RvStatus RvSipStackGetAuthenticatorHandle(
    IN RvSipStackHandle hStack,
    OUT RvSipAuthenticatorHandle *phAuth);
```

**PARAMETERS**

- **hStack**
  The handle to the SIP Stack.

- **phAuth**
  The handle to the *authenticator* handle.

**RETURN VALUES**

Returns *RvStatus*.
Stack Manager Get and Set Functions
RvSipStackGetTransportMgrHandle()

RvSipStackGetTransportMgrHandle()

DESCRIPTION

Gets the TransportMgr handle. You need this handle in order to use the Transport API functions and callbacks.

SYNTAX

RvStatus RvSipStackGetTransportMgrHandle(
    IN RvSipStackHandle hStack,
    OUT RvSipTransportMgrHandle *phTransportMgr);

PARAMETERS

hStack

The handle to the SIP Stack.

phTransportMg

The handle to the TransportMgr.

RETURN VALUES

Returns RvStatus.
RvSipStackGetSubsMgrHandle()

**DESCRIPTION**

Gets the SubscriptionMgr handle. You need this handle in order to use the Subscription API.

**SYNTAX**

```c
RvStatus RvSipStackGetSubsMgrHandle(
    IN RvSipStackHandle hStack,
    OUT RvSipSubsMgrHandle *phSubsMgr);
```

**PARAMETERS**

- **hStack**
  
  The handle to the SIP Stack.

- **phSubsMgr**
  
  The handle to the SubscriptionMgr.

**RETURN VALUES**

Returns RvStatus.
Stack Manager Get and Set Functions
RvSipStackGetTransmitterMgrHandle()

RvSipStackGetTransmitterMgrHandle()

DESCRIPTION
Gets the TransmitterMgr handle. You need this handle in order to use the Transmitter API functions.

SYNTAX
RvStatus RvSipStackGetTransmitterMgrHandle(
    IN  RvSipStackHandle hStack,
    OUT RvSipTransmitterMgrHandle  *phTrxMgr);

PARAMETERS
hStack
The handle to the SIP Stack.

phTrxMgr
The handle to the TransmitterMgr.

RETURN VALUES
Returns RvStatus.
RvSipStackGetResolverMgrHandle()

**DESCRIPTION**

Gets the ResolverMgr handle.

**SYNTAX**

```c
RvStatus RvSipStackGetResolverMgrHandle(
    IN RvSipStackHandle hStack,
    OUT RvSipResolverMgrHandle *phResolverMgr);
```

**PARAMETERS**

- **hStack**
  
  The handle to the SIP Stack.

- **phResolverMgr**
  
  The handle to the ResolverMgr.

**RETURN VALUES**

Returns RvStatus.
RvSipStackGetLogHandle()

DESCRIPTION
Gets the log handle to the SIP Stack object.

SYNTAX
RvStatus RvSipStackGetLogHandle(
    IN RvSipStackHandle hStack,
    OUT RV_LOG_Handle *phLog);

PARAMETERS

hStack
The handle to the SIP Stack.

phLog
The handle to the log.

RETURN VALUES
Returns RvStatus.
RvSipStackSetNewLogFilters()

DESCRIPTION
Sets the new log filters for the specified module in run-time. The new filters will replace the current filters the module is using. For example, they are a new set of filters that the module is going to use.

SYNTAX
RvStatus RvSipStackSetNewLogFilters(
    IN RvSipStackHandle hStack,
    IN RvSipStackModule module,
    IN RvUint8 filters);

PARAMETERS
hStack
The handle to the SIP Stack.

module
The module for which the filters are going to be changed.

filters
The new set of filters.

RETURN VALUES
Returns RvStatus.
Stack Manager Get and Set Functions
RvSipStackIsLogFilterExist()

RvSipStackIsLogFilterExist()

**DESCRIPTION**
Checks the existence of a filter for a specified module.

**SYNTAX**
RvBool RvSipStackIsLogFilterExist(
    IN RvSipStackHandle hStack,
    IN RvSipStackModule module,
    IN RvSipLogFilters filter);

**PARAMETERS**

*hStack*
The handle to the SIP Stack.

*module*
The module for which the existence of a filter is being checked.

*filter*
The filter being checked.

**RETURN VALUES**
Returns RV_TRUE if a filter exists for the specified module. Otherwise, the function returns RV_FALSE.
RvSipStackGetResources()  

**DESCRIPTION**  
Gets the status of resources used by the SIP Stack. The function is used only for the Call-leg, Message, Transaction, Register-Client and Stack Manager modules. You should specify the module name and give an appropriate structure. For example, if you wish to get the Register-Client resources, the module parameter should be set to RVSIP_REGCLIENT and the `pResources` parameter should be of type RvSipRegClientResources.

**SYNTAX**  
```
RvStatus RvSipStackGetResources(
    IN RvSipStackHandle hStack,
    IN RvSipStackModule module,
    OUT void *pResources);
```

**PARAMETERS**  
- **hStack**  
The handle to the SIP Stack object.
- **module**  
The module for which resource status is being checked.
- **pResources**  
The resources in use by the specified module.

**RETURN VALUES**  
Returns `RvStatus`.  

---
RvSipStackGetStatistics()

DESCRIPTION

Gets the statistics about the number of messages sent and received by the SIP Stack. You must supply a pointer to a valid RvSipStackStatistics structure. The SIP Stack will fill this structure with the current statistics.

SYNTAX

RvStatus RvSipStackGetStatistics(
    IN RvSipStackHandle hStack,
    OUT RvSipStackStatistics *pStatistics);

PARAMETERS

hStack
The handle to the SIP Stack.

pStatistics
The structure to be filled with the current statistics.

RETURN VALUES

Returns RvStatus.
RvSipStackMgrIsEnhancedDnsFeatureEnabled()  

**DESCRIPTION**  
Returns RV_TRUE if the SIP Stack was compiled with the RV_DNS_ENHANCED_FEATURES_SUPPORT flag.

**SYNTAX**  
RvBool RvSipStackMgrIsEnhancedDnsFeatureEnabled();

**PARAMETERS**  
None.

**RETURN VALUES**  
Returns RV_TRUE if the SIP Stack was compiled with the RV_DNS_ENHANCED_FEATURES_SUPPORT flag. Otherwise, returns RV_FALSE.
RvSipStackMgrIsTlsFeatureEnabled()

**DESCRIPTION**
Returns RV_TRUE if the SIP Stack was compiled with the RV_TLS_ON flag.

**SYNTAX**
RvBool RvSipStackMgrIsTlsFeatureEnabled();

**PARAMETERS**
None.

**RETURN VALUES**
Returns RV_TRUE if the SIP Stack was compiled with RV_TLS_ON flag. Otherwise, returns RV_FALSE.
RvSipStackIsSessionTimerSupported()

DESCRIPTION
Returns whether or not the SIP Stack supports the session-timer feature. The bIsSupported output parameter will be set to RV_TRUE if the SIP Stack supported list includes the “timer” option tag. Otherwise it will be set to RV_FALSE.

SYNTAX
RvStatus RvSipStackIsSessionTimerSupported(
    IN RvSipStackHandle hStack,
    OUT RvBool *bIsSupported);

PARAMETERS
hStack
The handle to the SIP Stack.

bIsSupported
RV_TRUE if session-timer is supported. Otherwise, RV_FALSE.

RETURN VALUES
Returns RvStatus.
RvSipStackIsReplacesSupported()

DESCRIPTION

Returns whether or not the SIP Stack supports the Replaces feature. The bIsSupported output parameter will be set to RV_TRUE if the SIP Stack supported list includes the “replaces” option tag. Otherwise it will be set to RV_FALSE.

SYNTAX

RvStatus RvSipStackIsReplacesSupported(
    IN  RvSipStackHandle    hStack,
    OUT RvBool          *pbIsSupported);

PARAMETERS

hCallLegMgr
The Call-legMgr handle.

pbIsSupported
RV_TRUE if Replaces is supported. Otherwise, RV_FALSE.

RETURN VALUES

Returns RvStatus.
The Call-leg API enables you to create calls, terminate calls, modify existing calls (re-INVITE) and also provides a set of callbacks that enable you to add your own call processing logic to the application.

This part includes the following section:

- Call-leg Functions
CALL-LEG FUNCTIONS

The Call-leg API functions of the SIP Stack enable you to create and manage call-leg objects (*call-legs*), connect and disconnect calls, transfer, authenticate, send different requests within a call, and control *call-leg* parameters. This section contains the Call-leg API functions included in the *RvSipCallLeg.h* header file.

The functions included in this section are:

- Call-leg Manager Functions
- Call-leg Session Functions
- Call-leg Get and Set Functions
- Call-leg High Availability Functions
- Call-leg Server Authentication Functions
- Call-leg Replaces Functions
- Call-leg Session-Timer Functions
- Call-leg DNS Functions
- Call-leg Local and Remote Address Functions
- Call-leg Forking Functions
The Call-leg Manager (Call-legMgr) is in charge of all the call-legs. It is used to set the event handlers of the Call-leg module and to create new call-legs.

The Call-leg Manager API functions are:

- `RvSipCallLegMgrSetEvHandlers()`
- `RvSipCallLegMgrCreateCallLeg()`
- `RvSipCallLegMgrSetAppMgrHandle()`
- `RvSipCallLegMgrGetAppMgrHandle()`
- `RvSipCallLegMgrGetStackInstance()`
RvSipCallLegMgrSetEvHandlers()

**DESCRIPTION**
Sets event handlers for all *call-leg* events.

**SYNTAX**

```c
RvStatus RvSipCallLegMgrSetEvHandlers(
    IN RvSipCallLegMgrHandle hMgr,
    IN RvSipCallLegEvHandlers *pEvHandlers,
    IN RvInt32 structSize);
```

**PARAMETERS**

- **hMgr**
  The handle to the *Call-legMgr*.

- **pEvHandlers**
  A pointer to the structure containing application event handler pointers.

- **structSize**
  The size of the event handler structure.

**RETURN VALUES**

Returns *RvStatus*. 
RvSipCallLegMgrCreateCallLeg()

DESCRIPTION

Creates a new outgoing *call-leg* and exchange handles with the application. The new *call-leg* assumes the IDLE state.

**To establish a new session**

1. Create a new *call-leg* with this function.
2. Set the To and From headers, at least.
3. Call the RvSipCallLegConnect() function.

SYNTAX

```c
RvStatus RvSipCallLegMgrCreateCallLeg(
    IN RvSipCallLegMgrHandle hCallLegMgr,
    IN RvSipAppCallLegHandle hAppCallLeg,
    OUT RvSipCallLegHandle *hCallLeg);
```

PARAMETERS

- **hCallLegMgr**
  
The handle to the *Call-legMgr*.

- **hAppCallLeg**
  
The application handle to the newly created *call-leg*.

- **RvSipCallLegHandle**
  
  SIP Stack handle to the *call-leg*.

RETURN VALUES

Returns RvStatus.
RvSipCallLegMgrSetAppMgrHandle()

DESCRIPTION
Saves the Call-legMgr handle to the application in the SIP Stack Call-legMgr.

SYNTAX
RvStatus RvSipCallLegMgrSetAppMgrHandle(
    IN RvSipCallLegMgrHandle hCallLegMgr,
    IN void* pAppCallLegMgr);

PARAMETERS
hCallLegMgr
The handle to the SIP Stack Call-legMgr.

pAppCallLegMgr
The application Call-legMgr handle.

RETURN VALUES
Returns RvStatus.
Call-leg Manager Functions
RvSipCallLegMgrGetAppMgrHandle()

RvSipCallLegMgrGetAppMgrHandle()

DESCRIPTION
RvSipCallLegMgrGetAppMgrHandle() returns the handle to the application Call-legMgr. You set this handle in the SIP Stack using the RvSipCallLegMgrSetAppMgrHandle() function.

SYNTAX
RvStatus RvSipCallLegMgrGetAppMgrHandle(
    IN RvSipCallLegMgrHandle hCallLegMgr,
    OUT void** pAppCallLegMgr);

PARAMETERS

hCallLegMgr
The handle to the SIP Stack Call-legMgr.

pAppCallLegMgr
The application Call-legMgr handle.

RETURN VALUES
RvStatus
Returns RvStatus.
RvSipCallLegMgrGetStackInstance()

DESCRIPTION
Returns the handle to the Stack instance to which this Call-legMgr belongs.

SYNTAX
RvStatus RvSipCallLegMgrGetStackInstance(
    IN RvSipCallLegMgrHandle hCallLegMgr,
    OUT void* *phStackInstance);

PARAMETERS

hCallLegMgr
The handle to the Stack Call-legMgr.

phStackInstance
A valid pointer which will be updated with a handle to the Stack instance.

RETURN VALUES
Returns RvStatus.
Call-leg Session Functions

CALL-LEG SESSION FUNCTIONS

A call-leg represents a SIP call-leg as defined in RFC 3261. This means that a call-leg is defined using the Call-ID, From and To headers. Using the Call-leg API, the user can initiate calls, react to incoming calls and disconnect calls.

The Call-leg Session functions are:

- RvSipCallLegMake()
- RvSipCallLegConnect()
- RvSipCallLegAccept()
- RvSipCallLegAuthenticate()
- RvSipCallLegReject()
- RvSipCallLegByeAccept()
- RvSipCallLegByeReject()
- RvSipCallLegProvisionalResponse()
- RvSipCallLegProvisionalResponseReliable()
- RvSipCallLegDisconnect()
- RvSipCallLegTranscCreate()
- RvSipCallLegTranscRequest()
- RvSipCallLegTranscResponse()
- RvSipCallLegTranscTerminate()
- RvSipCallLegTerminate()
- RvSipCallLegCancel()
- RvSipCallLegAck()
- RvSipCallLegDetachOwner()
- RvSipCallLegSendPrack()
- RvSipCallLegSendPrackResponse()
- RvSipCallLegReInviteCreate()
- RvSipCallLegReInviteSetAppHandle()
- RvSipCallLegReInviteRequest()
- RvSipCallLegReInviteAck()
- RvSipCallLegReInviteTerminate()
- RvSipCallLegReInviteGetCurrentState()
RvSipCallLegMake()

DESCRIPTION
Sets the To and From headers in the call-leg and initiates an outgoing call. This function causes an INVITE to be sent and the Call-leg state machine to progress to the INVITING state.

SYNTAX
RvStatus RvSipCallLegMake(
    IN RvSipCallLegHandle hCallLeg,
    IN RvChar* strFrom,
    IN RvChar* strTo);

PARAMETERS

hCallLeg
The handle to the call-leg the user wishes to connect.

strFrom
The string containing the From party header. For example, “From:SIP:172.20.1.1:5060”.

strTo
The string containing the To party header. For example, “To:SIP:172.20.5.5:5060”.

RETURN VALUES
Returns RvStatus.
RvSipCallLegConnect()

DESCRIPTION

Initiates an outgoing call. This function may be called only after the To and From fields were set. Calling this function causes an INVITE to be sent out and the Call-leg state machine to progress to the INVITING state.

SYNTAX

RvStatus RvSipCallLegConnect(
       IN RvSipCallLegHandle hCallLeg);

PARAMETERS

hCallLeg
The handle to the call-leg the user wishes to connect.

RETURN VALUES

Returns RvStatus.
RvSipCallLegAccept()

DESCRIPTION
Called by the application to indicate that it is willing to accept an incoming call or an incoming re-INVITE.

SYNTAX
RvStatus RvSipCallLegAccept(
    IN RvSipCallLegHandle hCallLeg);

PARAMETERS
hCallLeg
The handle to the call-leg the user wishes to accept.

RETURN VALUES
Returns RvStatus.
RvSipCallLegAuthenticate()

DESCRIPTION
Sends a request with authentication information. When a call-leg receives a 401 or 407 response indicating that a request was unauthenticated by the server or proxy, the call-leg assumes the UNAUTHENTICATED state. You may use RvSipCallLegAuthenticate() in the UNAUTHENTICATED state to re-send the request with authentication information. You can use this function to authenticate both the INVITE and BYE requests.

SYNTAX
RvStatus RvSipCallLegAuthenticate(
    IN RvSipCallLegHandle hCallLeg);

PARAMETERS
hCallLeg
The handle to the call-leg the user wishes to authenticate.

RETURN VALUES
Returns RvStatus.
RvSipCallLegReject()

DESCRIPTION
Can be used in theOFFERING state to reject an incoming call. This function can also be used to reject a modify (re-INVITE) request received by a connected call-leg.

SYNTAX
RvStatus RvSipCallLegReject(
    IN RvSipCallLegHandle hCallLeg,
    IN RvUint16 status);

PARAMETERS

hCallLeg
The handle to the call-leg the user wishes to reject.

status
The rejection response code.

RETURN VALUES
Returns RvStatus.
Call-leg Session Functions
RvSipCallLegByeAccept()

---

**RvSipCallLegByeAccept()**

**DESCRIPTION**
Called by the application to indicate that it is willing to accept an incoming BYE request. By default, BYE requests are responded to automatically with 200 OK. If the application wishes to respond differently, it should register to the RvSipCallLegByeCreatedEv() and RvSipCallLegByeStateChangedEv() callbacks.

When a BYE is received, the application will be informed at the RVSIP_CALL_LEG_BYE_REQUEST_RCVD state. The application should use the RvSipCallLegByeAccept() function to accept the BYE request.

**SYNTAX**

```c
RvStatus RvSipCallLegByeAccept(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransc);
```

**PARAMETERS**

- **hCallLeg**
  The handle to the *call-leg* whose BYE the user wishes to accept.

- **hTransc**
  The handle to the *BYE transaction*.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegByeReject()

DESCRIPTION

Rejects a BYE request. By default, BYE requests are responded to automatically with 200 OK. If the application wishes to respond differently, it should register to the RvSipCallLegByeCreatedEv() and RvSipCallLegByeStateChangedEv() callbacks.

When a BYE is received, the application will be informed at the RVSIP_CALL_LEG_BYE_REQUEST_RCVD state. The application should use the RvSipCallLegByeReject() function to accept the BYE request.

SYNTAX

RvStatus RvSipCallLegByeReject(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransc,
    IN RvUint16 status);

PARAMETERS

hCallLeg
The handle to the call-leg the user wishes to reject.

hTransc
The handle to the BYE transaction.

status
The rejection response code.

RETURN VALUES

Returns RvStatus.
**RvSipCallLegProvisionalResponse()**

**DESCRIPTION**

Sends a provisional response (1xx class) to the remote party. This function can be called whenever a request is received, for example, in the OFFERING state.

**SYNTAX**

```c
RvStatus RvSipCallLegProvisionalResponse(
    IN RvSipCallLegHandle hCallLeg,
    IN RvUint16 status);
```

**PARAMETERS**

- **hCallLeg**
  
  The handle to the *call-leg*.

- **status**

  The provisional response status code.

**RETURN VALUES**

Returns `RvStatus`. 
RvSipCallLegProvisionalResponseReliable()

DESCRIPTION

Sends a reliable provisional response (1xx class other than 100) to the remote party. This function can be called when an INVITE request is received, for example, in the OFFERING state.

SYNTAX

RvStatus RvSipCallLegProvisionalResponseReliable(
    IN RvSipCallLegHandle hCallLeg,
    IN RvUint16 status);

PARAMETERS

hCallLeg

The handle to the call-leg.

status

The provisional response status code.

RETURN VALUES

Returns RvStatus.
RvSipCallLegDisconnect()

DESCRIPTION

Causes the call to disconnect. RvSipCallLegDisconnect() may be called in any state. The behavior of the function depends on the following call-leg states:

- INVITING—CANCEL is sent and the call moves to the CANCELLING state.
- CONNECTED and ACCEPTED—BYE is sent and the call moves to the DISCONNECTING state.
- OFFERING—The incoming INVITE is rejected with status code 403.
- IDLE, DISCONNECTING, DISCONNECTED, UNAUTHENTICATED, REDIRECTED—The call is terminated.

If the function fails to send the BYE request, the call-leg will be terminated.

SYNTAX

RvStatus RvSipCallLegDisconnect(
    IN RvSipCallLegHandle hCallLeg);

PARAMETERS

hCallLeg

The handle to the call-leg the user wishes to disconnect.

RETURN VALUES

Returns RvStatus.
RvSipCallLegTranscCreate()

DESCRIPTION

Creates a new general transaction that is related to the supplied call-leg. The transaction will have call-leg characteristics such as To header, From header, Call-ID, and local and outbound addresses. The application can define an application handle to the transaction and supply it to the SIP Stack when calling this function. The application handle will be supplied back to the application when the RvSipCallLegTranscStateChangedEv() callback is called.

SYNTAX

RvStatus RvSipCallLegTranscCreate(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppTranscHandle hAppTransc,
    OUT RvSipTranscHandle *hTransc);

PARAMETERS

hCallLeg
The handle to the call-leg that will send the new request.

hAppTransc
The application handle to this transaction.

hTransc
The handle to the newly created transaction

RETURN VALUES

Returns RvStatus.
RvSipCallLegTranscRequest()

**DESCRIPTION**

Sends a request message with a given method using a given transaction. You can use this function in any call-leg state for sending requests, such as INFO. The request will have the To header, From header and Call-ID of the call-leg, and will be sent with a correct CSeq step. The request will be record-routed if needed.

**Note** Before calling this function, you should create a new call-leg transaction using the RvSipCallLegTranscCreate() function. You can then use the transaction outbound message mechanism to add headers and a body to the outgoing request. If you supply the function with a NULL transaction, the SIP Stack will create a new call-leg transaction automatically. In this case you will not be able to use the transaction outbound message.

**Note** A transaction that was supplied by the application will not be terminated if this function fails. It is the responsibility of the application to terminate the transaction using the RvSipCallLegTranscTerminate() function.

**SYNTAX**

RvStatus RvSipCallLegTranscRequest(
    IN RvSipCallLegHandle hCallLeg,
    IN RvChar* strMethod,
    OUT RvSipTranscHandle *hTransc);

**PARAMETERS**

**hCallLeg**

The handle to the call-leg that will send the new Request message.

**strMethod**

A string with the request method.
**hTransc**

The handle to the newly created *transaction*.

**RETURN VALUES**

Returns *RvStatus*. 
RvSipCallLegTranscResponse()

**DESCRIPTION**

Sends a response to a call-leg related transaction. When a call-leg receives a general request, such as INFO (but not BYE or PRACK), the call-leg first notifies the application that a new call-leg transaction was created using the RvSipCallLegTranscCreatedEv() callback. At this stage, the application can specify whether or not it wishes to handle the transaction, and the application can also replace handles with the SIP Stack. The call-leg will then notify the application about the new transaction state, GENERAL_REQUEST_RCVD, using the RvSipCallLegTranscStateChangedEv() callback. In this state, the application should use the RvSipCallLegTranscResponse() function to send a response to the request.

**SYNTAX**

RvStatus RvSipCallLegTranscResponse(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransc,
    IN RvUint16 status);

**PARAMETERS**

**hCallLeg**
The handle to the call-leg that will send the response message.

**hTrans**
The handle to the transaction.

**status**
The response code that is sent in the response message.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegTranscTerminate()

DESCRIPTION
Terminates a transaction related to a specified call-leg. You can use this function only for call-leg general transactions that are handled by the application. You can also use this function to terminate a BYE transaction in case you registered to the RvSipCallLegByeCreatedEv() and RvSipCallLegByeStateChangedEv() callbacks.

SYNTAX
RvStatus RvSipCallLegTranscTerminate(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransc);

PARAMETERS
hCallLeg
The handle to the call-leg to which the terminated transaction relates.

hTransc
The handle to the transaction that is terminated.

RETURN VALUES
Returns RvStatus.
**RvSipCallLegTerminate()**

**DESCRIPTION**
Terminates a *call-leg* without sending any messages (CANCEL or BYE). The *call-leg* will assume the TERMINATED state. Calling this function will cause an abnormal termination. All *transactions* related to the *call-leg* will be terminated as well.

**SYNTAX**

```c
RvStatus RvSipCallLegTerminate(
    IN RvSipCallLegHandle hCallLeg);
```

**PARAMETERS**

- **hCallLeg**
  The handle to the *call-leg* the user wishes to terminate.

**RETURN VALUES**

Returns `RvStatus`.
**RvSipCallLegCancel()**

**DESCRIPTION**

Cancels an INVITE request (or re-INVITE request). Calling this function causes a CANCEL message to be sent to the remote party. You can call this function in the following *call-leg* states:

- RVSIP_CALL_LEG_STATE_INVITING
- RVSIP_CALL_LEG_STATE_PROCEEDING
- PROCEEDING_TIMEOUT

and in the following MODIFY states:

- RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_SENT
- RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_PROCEEDING
- RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_PROCEEDING_TIMEOUT

**SYNTAX**

```c
RvStatus RvSipCallLegCancel(
    IN RvSipCallLegHandle hCallLeg);
```

**PARAMETERS**

- **hCallLeg**
  
The handle to the *call-leg* that the user wishes to cancel.

**RETURN VALUES**

Returns `RvStatus`.
## RvSipCallLegAck()

### DESCRIPTION

Sends an ACK request from the *call-leg* to the remote party. When the SIP Stack is configured to work in a manual ACK mode, the *call-leg* will not send the ACK message after receiving a 2xx response. The application should use the RvSipCallLegAck() function to trigger the *call-leg* to send the ACK. This function can be called only in the REMOTE_ACCEPTED state. It sends an ACK only for the initial INVITE request. You can use this function for re-INVITE only if you configured the SIP Stack to work with the “old INVITE” behavior (according to *bis-2543*).

### SYNTAX

```c
RvStatus RvSipCallLegAck(
    IN RvSipCallLegHandle hCallLeg);
```

### PARAMETERS

**hCallLeg**

The handle to the *call-leg*.

### RETURN VALUES

Returns RvStatus.
RvSipCallLegDetachOwner()

DESCRIPTION
Detaches the call-leg owner. After calling this function, the user will stop receiving events for this call-leg. This function can be called only after calling the RvSipCallLegTerminate() function. This will prevent the call-leg from notifying about the TERMINATED state.

SYNTAX
RvStatus RvSipCallLegDetachOwner(
    IN RvSipCallLegHandle hCallLeg);

PARAMETERS
hCallLeg
The handle to the call-leg whose owner the user wishes to detach.

RETURN VALUES
Returns RvStatus.
**RvSipCallLegSendPrack()**

**DESCRIPTION**

When the SIP Stack is configured to work in a manual PRACK mode, the application is responsible for generating the PRACK message whenever a reliable provisional response is received. When a reliable provisional response is received, the Call-leg PRACK state machine assumes the REL_PROV_RESPONSE_RCVD state. You should then call the RvSipCallLegSendPrack() function to send the PRACK message to the remote party. The Call-leg PRACK state machine will then assume the PRACK_SENT state.

**Note**  The SIP Stack is responsible for adding the RAck header to the PRACK message.

**SYNTAX**

```c
RvStatus RvSipCallLegSendPrack(
    IN RvSipCallLegHandle hCallLeg);
```

**PARAMETERS**

`hCallLeg`

The *transaction* handle.

**RETURN VALUES**

Returns `RvStatus`. 
RvSipCallLegSendPrackResponse()

DESCRIPTION
When the SIP Stack is configured to work in a manual PRACK mode, the application is responsible for responding to any PRACK request that is received for a previously-sent reliable provisional response. When a PRACK request is received, the Call-leg PRACK state machine assumes the PRACK_RCVD state. You should then call the RvSipCallLegSendPrackResponse() function to send a response to the PRACK request. The Call-leg PRACK state machine will then assume the PRACK_FINAL_RESPONSE_SENT state.

SYNTAX
RvStatus RvSipCallLegSendPrackResponse(
        IN RvSipCallLegHandle hCallLeg,
        IN RvUint16 responseCode);

PARAMETERS
hCallLeg
The transaction handle.

responseCode
The response code to send.

RETURN VALUES
Returns RvStatus.
RvSipCallLegReInviteCreate()

DESCRIPTION

Creates a new re-INVITE object in a call-leg. A re-INVITE object may be created in the following situations:

- The call-leg had already sent/received the 2xx response for the initial INVITE request (the call-leg state is CONNECTED, ACCEPTED or REMOTE-ACCEPTED).
- There are no other pending re-INVITE transactions (no other re-Invite objects that are waiting for a final response).

SYNTAX

RvStatus RvSipCallLegReInviteCreate(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegInviteHandle hAppReInvite,
    OUT RvSipCallLegInviteHandle* phReInvite);

PARAMETERS

hCallLeg
The handle to the call-leg that the user wishes to modify.

hAppReInvite
The application handle to the new re-Invite object.

phReInvite
A pointer to the new re-Invite object.

RETURN VALUES

Returns RvStatus.
RvSipCallLegReInviteSetAppHandle()  

**DESCRIPTION**  
Sets the re-INVITE application handle. Usually the application replaces handles with the Stack in the RvSipCallLegReInviteCreatedEv() callback or the RvSipCallLegReInviteCreate() function. This function is used if the application wishes to set a new application handle.

**SYNTAX**  
RvStatus RvSipCallLegReInviteSetAppHandle (  
  IN RvSipCallLegHandle hCallLeg,  
  IN RvSipCallLegInviteHandle hReInvite,  
  IN RvSipAppCallLegInviteHandle hAppReInvite)

**PARAMETERS**  

**hCallLeg**  
The handle to the call-leg.

**hReInvite**  
The handle to the re-INVITE object.

**hAppReInvite**  
The handle to the application re-INVITE object.

**RETURN VALUES**  
Returns RvStatus.
RvSipCallLegReInviteRequest()

**DESCRIPTION**
Sends a re-INVITE request from a *call-leg* to the remote party.

**Note**  This function does not refer to the parameters of the session-timer *call-leg*. Thus, when used during a session-timer call, it turns off the session-timer mechanism. Consequently, to keep the mechanism up, use the RvSipCallLegSessionTimerInviteRefresh() function instead.

**SYNTAX**

```c
RvStatus RvSipCallLegReInviteRequest(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipCallLegInviteHandle hReInvite);
```

**PARAMETERS**

**hCallLeg**
The handle to the *call-leg* that the user wishes to modify.

**hReInvite**
The handle to the re-INVITE object.

**RETURN VALUES**
Returns RvStatus.
**RvSipCallLegReInviteAck()**

**DESCRIPTION**
Sends an ACK request for 2xx response on re-Invite. When the SIP Stack is configured to work in manual ACK mode, the *call-leg* will not send the ACK message after receiving a 2xx response by itself. The application should use this function to trigger the *call-leg* to send the ACK. This function can be called only in the MODIFY_REINVITE_REMOTE_ACCEPTED state.

**SYNTAX**

```c
RvStatus RvSipCallLegReInviteAck(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipCallLegInviteHandle hReInvite);
```

**PARAMETERS**

- **hCallLeg**
  The handle to the *call-leg*.

- **hReInvite**
  The handle to the re-INVITE object.

**RETURN VALUES**
Returns RvStatus.
Call-leg Session Functions
RvSipCallLegReInviteTerminate()  

RvSipCallLegReInviteTerminate()

DESCRIPTION
Terminates a re-INVITE object without sending any messages (CANCEL or BYE). The re-INVITE object will assume the TERMINATED state.

SYNTAX
RvStatus RvSipCallLegReInviteTerminate(  
    IN RvSipCallLegHandle hCallLeg,  
    IN RvSipCallLegInviteHandle hReInvite);

PARAMETERS

hCallLeg
The handle to the call-leg.

hReInvite
The handle to the re-INVITE object.

RETURN VALUES
Returns RvStatus.
RvSipCallLegReInviteGetCurrentState()

**DESCRIPTION**
Gets the current state of the re-INVITE of the call-leg.

**SYNTAX**

```c
RvStatus RvSipCallLegReInviteGetCurrentState(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipCallLegInviteHandle hReInvite,
    OUT RvSipCallLegModifyState *peState);
```

**PARAMETERS**

- **hCallLeg**
  The handle to the call-leg.

- **hReInvite**
  The handle to the re-INVITE object.

- **peState**
  The current state of the re-INVITE object.

**RETURN VALUES**

Returns RvStatus.
CALL-LEG GET AND SET FUNCTIONS

The Call-leg API provides functions to set and access the *call-leg* fields.

The Call-leg Get and Set functions are:

- RvSipCallLegGetNewMsgElementHandle()
- RvSipCallLegSetCallId()
- RvSipCallLegGetCallId()
- RvSipCallLegSetFromHeader()
- RvSipCallLegGetFromHeader()
- RvSipCallLegSetToHeader()
- RvSipCallLegGetToHeader()
- RvSipCallLegSetRemoteContactAddress()
- RvSipCallLegGetRemoteContactAddress()
- RvSipCallLegSetLocalContactAddress()
- RvSipCallLegGetLocalContactAddress()
- RvSipCallLegGetReceivedFromAddress()
- RvSipCallLegSetCSeq()
- RvSipCallLegGetCSeq()
- RvSipCallLegGetRemoteCSeq()
- RvSipCallLegSetAppHandle()
- RvSipCallLegGetAppHandle()
- RvSipCallLegGetCurrentState()
- RvSipCallLegGetDirection()
- RvSipCallLegGetTranscByMsg()
- RvSipCallLegGet100RelStatus()
- RvSipCallLegGetCallLegMgr()
- RvSipCallLegGetReplacesStatus()
- RvSipCallLegGetReplacesHeader()
- RvSipCallLegSetReplacesHeader()
- RvSipCallLegGetReceivedMsg()
- RvSipCallLegGetOutboundMsg()
- RvSipCallLegResetOutboundMsg()
- RvSipCallLegSetOutboundMsg()
- RvSipCallLegGetSubscription()
- RvSipCallLegSetPersistency()
Call-leg Get and Set Functions

- RvSipCallLegGetPersistency()
- RvSipCallLegSetConnection()
- RvSipCallLegGetConnection()
- RvSipCallLegGetStackInstance()
- RvSipCallLegSetRejectStatusCodeOnCreation()
- RvSipCallLegUseFirstRouteForInitialRequest()
- RvSipCallLegSetForceOutboundAddrFlag()
- RvSipCallLegSetAddAuthInfoToMsgFlag()
- RvSipCallLegSetTranscTimers()
RvSipCallLegGetNewMsgElementHandle()

**DESCRIPTION**

Allocates a new header on the *call-leg* page, and returns the new header handle. The application may use this function to allocate a *message* header or a *message* address. It should then fill the element information, and set it back to the *call-leg* using the relevant Set function. The function supports the following headers:

- **Party**—you should set these headers back with `RvSipCallLegSetToHeader()` or `RvSipCallLegSetFromHeader()`.
- **Replaces**—you should set these headers back with `RvSipCallLegSetReplacesHeader()`.
- **Authorization**—you should set these headers back with `RvSipCallLegSetInitialAuthorization()` (for IMS only).
- **Address**—you should set these headers back with `RvSipCallLegSetRemoteContactAddress()` or `RvSipCallLegSetLocalContactAddress()`.

**Note** You may use this function only in the INITIAL state (before sending a message). In any other state you must construct the header on an application page, and then set it to the SIP Stack object.

**SYNTAX**

```c
RvStatus RvSipCallLegGetNewMsgElementHandle {
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipHeaderType eHeaderType,
    IN RvSipAddressType eAddrType,
    OUT void* *phHeader;
};
```

**PARAMETERS**

- **hCallLeg**
  
  The handle to the *call-leg*.
eHeaderType
The type of header to allocate. RVSIP_HEADERTYPE_UNDEFINED when allocating an address.

eAddrType
The type of the address in case of address allocation.

phHeader
The handle to the newly created header or address.

**RETURN VALUES**
Returns RvStatus.
**RvSipCallLegSetCallId()**

**DESCRIPTION**
Sets the call-leg Call-ID.

**SYNTAX**

```c
RvStatus RvSipCallLegSetCallId(
    IN RvSipCallLegHandle hCallLeg,
    IN RvChar *strCallId);
```

**PARAMETERS**

- **hCallLeg**
The SIP Stack handle to the call-leg.

- **strCallId**
A NULL terminating string with the new Call-ID.

**RETURN VALUES**
Returns RvStatus.
RvSipCallLegGetCallId()

DESCRIPTION
Returns the call-leg Call-ID.

SYNTAX
RvStatus RvSipCallLegGetCallId(
    IN RvSipCallLegHandle hCallLeg,
    IN RvInt32 bufSize,
    OUT RvChar *strCallId,
    OUT RvInt32 *actualSize);

PARAMETERS

hCallLeg
The SIP Stack handle to the call-leg.

bufSize
The size of the application buffer for the Call-ID.

strCallId
An application allocated buffer.

actualSize
The actual size of the Call-ID.

RETURN VALUES
Returns RvStatus.

REMARKS
If the buffer allocated by the application is insufficient an RV_ERROR_INSUFFICIENT_BUFFER status is returned and actualSize contains the size of the Call-ID string in the call-leg.
RvSipCallLegSetFromHeader()

DESCRIPTION
Sets the From header associated with the call-leg. Attempting to alter the From header after the call has been initiated may cause unexpected results.

SYNTAX
RvStatus RvSipCallLegSetFromHeader(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipPartyHeaderHandle hFrom);

PARAMETERS

hCallLeg
The handle to the call-leg.

hFrom
The handle to an application-constructed From header.

RETURN VALUES
Returns RvStatus.
RvSipCallLegGetFromHeader()

DESCRIPTION
Returns the From header associated with a call-leg. Attempting to alter the From header after the call has been initiated might cause unexpected results.

SYNTAX
RvStatus RvSipCallLegGetFromHeader(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipPartyHeaderHandle *phFrom);

PARAMETERS
hCallLeg
The handle to the call-leg.

phFrom
A pointer to the call-leg From header handle.

RETURN VALUES
Returns RvStatus.
RvSipCallLegSetToHeader()

DESCRIPTION
Sets the To header associated with a call-leg. Attempting to alter the To header after the call has been initiated may cause unexpected results.

SYNTAX
RvStatus RvSipCallLegSetToHeader (  
    IN RvSipCallLegHandle hCallLeg,  
    IN RvSipPartyHeaderHandle hTo);  

PARAMETERS
hCallLeg
The handle to the call-leg.

hTo
The handle to an application-constructed To header.

RETURN VALUES
Returns RvStatus.
RvSipCallLegGetToHeader()

DESCRIPTION
Returns the To header associated with a call-leg. Attempting to alter the To address after a call has been initiated may cause unexpected results.

SYNTAX
RvStatus RvSipCallLegGetToHeader(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipPartyHeaderHandle *phTo);

PARAMETERS
hCallLeg
The handle to the call-leg.

phTo
A pointer to the call-leg To header handle.

RETURN VALUES
Returns RvStatus.
RvSipCallLegSetRemoteContactAddress()

DESCRIPTION
Sets the contact address of the remote party. This is the address with which the remote party may be contacted. This function may be used for outgoing calls when the user wishes to use a Request-URI that is different from the To header of the call-leg. You should set the remote contact address only in the IDLE state.

SYNTAX
RvStatus RvSipCallLegSetRemoteContactAddress(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAddressHandle hContactAddress);

PARAMETERS
hCallLeg
The handle to the call-leg.

hContactAddress
The handle to the Address header.

RETURN VALUES
Returns RvStatus.
RvSipCallLegGetRemoteContactAddress()

**DESCRIPTION**

Gets the contact address of the remote party. This is the address the remote party supplies for contact in future requests.

**SYNTAX**

RvStatus RvSipCallLegGetRemoteContactAddress(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipAddressHandle *phContactAddress);

**PARAMETERS**

- **hCallLeg**
  The handle to the call-leg.

- **phContactAddress**
  A pointer to the remote contact address of the call-leg.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegSetLocalContactAddress()

DESCRIPTION
Sets the local contact address that the SIP Stack uses to identify itself to the remote party. The remote party uses the contact address to directly contact the local party.

SYNTAX
RvStatus RvSipCallLegSetLocalContactAddress(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAddressHandle hContactAddress);

PARAMETERS

hCallLeg
The handle to the call-leg.

hContactAddress
The handle to the local contact address header.

RETURN VALUES
Returns RvStatus.
RvSipCallLegGetLocalContactAddress()

DESCRIPTION
Gets the local contact address with which the SIP Stack identifies itself to the remote party. If no value is supplied, the From header address part is taken. The remote party uses the local contact address to directly contact the local party.

SYNTAX
RvStatus RvSipCallLegGetLocalContactAddress(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipAddressHandle *phContactAddress);

PARAMETERS

hCallLeg
The handle to the call-leg.

phContactAddress
The handle to the local contact address header.

RETURN VALUES
Returns RvStatus.
RvSipCallLegGetReceivedFromAddress()

DESCRIPTION
Gets the address from which the last message was received.

SYNTAX
RV_Status RvSipCallLegGetReceivedFromAddress(
    IN   RvSipCallLegHandle   hCallLeg,
    OUT  RvSipTransportAddr   *pAddr);

PARAMETERS
hCallLeg
The call-leg handle.

pAddr
Basic details about the received From address.

RETURN VALUES
Returns RvStatus.
RvSipCallLegSetCSeq()

**DESCRIPTION**
Sets the outgoing *call-leg* CSeq step counter.

**SYNTAX**

```
RvStatus RvSipCallLegSetCSeq(
    IN RvSipCallLegHandle hCallLeg,
    IN RvInt32 cseq);
```

**PARAMETERS**

- **hCallLeg**
  The SIP Stack handle to the *call-leg*.

- **cseq**
  The Cseq step.

**RETURN VALUES**

Returns RvStatus.
Call-leg Get and Set Functions

RvSipCallLegGetCSeq()

RvSipCallLegGetCSeq()

DESCRIPTION
Gets the outgoing call-leg CSeq step counter.

SYNTAX
RvStatus RvSipCallLegGetCSeq(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvInt32 *cseq);

PARAMETERS

hCallLeg
The SIP Stack handle to the call-leg.

cseq
The Cseq step.

RETURN VALUES
Returns RvStatus.
RvSipCallLegGetRemoteCSeq()

**DESCRIPTION**
Gets the incoming *call-leg* CSeq step counter.

**SYNTAX**

```c
RvStatus RvSipCallLegGetRemoteCSeq(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvInt32 *cseq);
```

**PARAMETERS**

- **hCallLeg**
  The SIP Stack handle to the *call-leg*.

- **cseq**
  The remote CSeq step.

**RETURN VALUES**

Returns `RvStatus`.
**RvSipCallLegSetAppHandle()**

**DESCRIPTION**
Sets the *call-leg* application handle. Usually the application replaces handles with the SIP Stack in the `RvSipCallLegCreatedEv()` callback or the `RvSipCallLegMgrCreateCallLeg()` API function. This function is used if the application wishes to set a new application handle.

**SYNTAX**

```c
RvStatus RvSipCallLegSetAppHandle(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg);
```

**PARAMETERS**

- **hCallLeg**
  The handle to the *call-leg*.

- **hAppCallLeg**
  The new application handle to the *call-leg*.

**RETURN VALUES**

Returns `RvStatus`. 
RvSipCallLegGetAppHandle()

DESCRIPTION
Returns the application handle of this call-leg.

SYNTAX
RvStatus RvSipCallLegGetAppHandle (  
    IN  RvSipCallLegHandle hCallLeg,  
    OUT RvSipAppCallLegHandle *phAppCallLeg);

PARAMETERS
hCallLeg
The handle to the call-leg.

phAppCallLeg
The application handle to the call-leg.

RETURN VALUES
Returns RvStatus.
RvSipCallLegGetCurrentState()

**DESCRIPTION**

Gets the current state of the call-leg.

**SYNTAX**

```c
RvStatus RvSipCallLegGetCurrentState(
    IN  RvSipCallLegHandle hCallLeg,
    OUT RvSipCallLegState  *peState);
```

**PARAMETERS**

- **hCallLeg**
  The call-leg handle.

- **peState**
  The current state of the call-leg.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegGetDirection()

DESCRIPTION
Queries the call-leg direction. A call-leg can be either an incoming or outgoing call. When you create a call, it is always an outgoing call. If the call is created because an INVITE has arrived, the call is an incoming call.

SYNTAX
RvStatus RvSipCallLegGetDirection(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipCallLegDirection *peDirection);

PARAMETERS
hCallLeg
The call-leg handle.

peDirection
The call-leg direction.

RETURN VALUES
Returns RvStatus.
Call-leg Get and Set Functions
RvSipCallLegGetTranscByMsg()

**RvSipCallLegGetTranscByMsg()**

**DESCRIPTION**

Gets the *transaction* that owns a given message. Use this function to find the *transaction* handle to a specific message.

**Note** You cannot get Cancel *transactions* with this function.

**SYNTAX**

```c
RvStatus RvSipCallLegGetTranscByMsg(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipMsgHandle hMsg,
    IN RvBool bIsMsgRcvd,
    OUT RvSipTranscHandle *phTransc);
```

**PARAMETERS**

- **hCallLeg**
  The *call-leg* handle.

- **hMsg**
  The message handle.

- **bIsMsgRcvd**
  RV_TRUE if this is a received message. Otherwise, RV_FALSE.

- **phTransc**
  The handle to the *transaction* to which this message belongs.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegGet100RelStatus()

DESCRIPTION
Returns the 100rel option tag status of a received INVITE request. The 100rel option tag indicates whether the remote party supports or requires the PRACK extension. In case of a Require status, the application should use the RvSipCallLegProvisionalResponseReliable() function and send a reliable provisional response.

SYNTAX
RvStatus RvSipCallLegGet100RelStatus(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipTransaction100RelStatus *relStatus);

PARAMETERS

hCallLeg
The transaction handle.

relStatus
The reliable status received in the INVITE request.

RETURN VALUES
Returns RvStatus.
Call-leg Get and Set Functions
RvSipCallLegGetCallLegMgr()

---

**RvSipCallLegGetCallLegMgr()**

**DESCRIPTION**
Returns the *Call-legMgr* handle.

**SYNTAX**

```c
RvStatus RvSipCallLegGetCallLegMgr(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipCallLegMgrHandle *phCallLegMgr);
```

**PARAMETERS**

- **hCallLeg**
  The *call-leg* handle.

- **phCallLegMgr**
  The *Call-legMgr* handle.

**RETURN VALUES**

Returns *RvStatus*. 
RvSipCallLegGetReplacesStatus()

DESCRIPTION
Returns the Replaces option tag status of a received Invite/REFER request.

SYNTAX
RvStatus RvSipCallLegGetReplacesStatus(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipCallLegReplacesStatus *replacesStatus);

PARAMETERS

hCallLeg
The call-leg handle.

replacesStatus
The Replaces status received in the Invite or REFER request.

RETURN VALUES
Returns RvStatus.
Call-leg Get and Set Functions
RvSipCallLegGetReplacesHeader()

**RvSipCallLegGetReplacesHeader()**

**DESCRIPTION**

Gets the Replaces header from a call-leg. This function should be called before sending an INVITE to get the Replaces header that the REFER message set, if it exists. Or the function should be called from the evStateChanged when receiving an Invite, to know if the INVITE contained a Replaces header.

**SYNTAX**

```c
RvStatus RvSipCallLegGetReplacesHeader(  
    IN RvSipCallLegHandle hCallLeg,  
    OUT RvSipReplacesHeaderHandle *phReplacesHeader);
```

**PARAMETERS**

- **hCallLeg**
  The handle to the call-leg.

- **hReplacesHeader**
  The returned Replaces header.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegSetReplacesHeader()

DESCRIPTION
Sets the Replaces header in a call-leg. This function can be called before sending the INVITE request, when the call is in the IDLE state. The application should call this function to add a Replaces header to the INVITE request. If the application does not want to add the Replaces header that was received in the REFER to the INVITE (that is triggered by the Refer), it should call this function with NULL as the Replaces header.

SYNTAX
RvStatus RvSipCallLegSetReplacesHeader(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipReplacesHeaderHandle hReplacesHeader);

PARAMETERS
hCallLeg
The handle to the call-leg.

hReplacesHeader
The handle to a Replaces header.

RETURN VALUES
Returns RvStatus.
RvSipCallLegGetReceivedMsg()

**DESCRIPTION**

Gets the message that was received by the call-leg. You can call this function from the state changed call back function, RvSipCallLegStateChangedEv(), when the new state indicates that a message was received. If there is no valid received message, NULL will be returned.

**SYNTAX**

```c
RvStatus RvSipCallLegGetReceivedMsg(
    IN  RvSipCallLegHandle hCallLeg,
    OUT RvSipMsgHandle   *phMsg);
```

**PARAMETERS**

- **hCallLeg**
  
The call-leg handle.

- **phMsg**
  
A pointer to the received message.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegGetOutboundMsg()

**DESCRIPTION**

Gets the message that is going to be sent by the call-leg. You can call this function before calling a function that causes a message to be sent, such as RvSipCallLegConnect(), RvSipCallLegAccept() and RvSipCallLegReject(). The function returns an empty message object. You can fill the message object with headers and a body and then call the function. The call-leg will add more headers to the same message object and the message will be sent.

**SYNTAX**

RvStatus RvSipCallLegGetOutboundMsg(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipMsgHandle *phMsg);

**PARAMETERS**

- **hCallLeg**
  The call-leg handle.

- **phMsg**
  A pointer to the message.

**RETURN VALUES**

Returns RvStatus.

**REMARKS**

- You must not use this function to add headers that are part of the call-leg or the transaction key such as To, From, Call-ID and Via. Doing so will cause unexpected behavior of the call-leg.
- If your application is multithreaded, you must lock your application object before calling the RvSipCallLegGetOutboundMsg() function and unlock it only after calling the function that sends the message.
RvSipCallLegResetOutboundMsg()

DESCRIPTION
Sets the outbound message of the call-leg to NULL. If the call-leg is about to send a message, it will create a new message object.

SYNTAX
RvStatus RvSipCallLegResetOutboundMsg(
    IN RvSipCallLegHandle hCallLeg);

PARAMETERS
hCallLeg
The call-leg handle.

RETURN VALUES
Returns RvStatus.
RvSipCallLegSetOutboundMsg()

**DESCRIPTION**
Copies an application-constructed message into the outbound message of the call-leg. It is the responsibility of the application to free this message after calling the Set function. It is not recommended to use this function and it is not documented.

**SYNTAX**

```c
RvStatus RvSipCallLegSetOutboundMsg(  
    IN RvSipCallLegHandle hCallLeg,  
    IN RvSipMsgHandle hMsg);
```

**PARAMETERS**

- **hCallLeg**
The call-leg handle.

- **hMsg**
The handle to the message.

**RETURN VALUES**
Returns RvStatus.
Call-leg Get and Set Functions
RvSipCallLegGetSubscription()

RvSipCallLegGetSubscription()

DESCRIPTION

Gets a subscription handle from call-leg subscriptions list. The user can use the location and hItem parameters to go over the list.

SYNTAX

RvStatus RvSipCallLegGetSubscription(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipListLocation location,
    IN OUT RvSipSubsHandle hItem,
    OUT RvSipSubsHandle *phSubs);

PARAMETERS

hCallLeg
The call-leg handle.

location
The location in list: next, previous, first or last.

hItem
The handle to the current position in the list (a relative subscription from the list). Supply this value if you choose next or previous in the location parameter.

phSubs
The handle to the returned subscription.

RETURN VALUES

Returns RvStatus.
RvSipCallLegSetPersistency()

**DESCRIPTION**
Changes the call-leg persistency definition at runtime. This function receives a Boolean value that indicates whether or not the application wishes this call-leg to be persistent. A persistent call-leg object will apply its persistency to the transactions it creates. A persistent transaction will try to locate a suitable connection in the connection hash before opening a new connection. A persistent call-leg will also try to use the same connection for all outgoing requests by applying the last-used connection to the next-created transaction.

**SYNTAX**
RvStatus RvSipCallLegSetPersistency(
    IN RvSipCallLegHandle hCallLeg,
    IN RvBool bIsPersistent);

**PARAMETERS**
- **hCallLeg**
The call-leg handle.
- **bIsPersistent**
Determines the call-leg persistency definition. RV_TRUE indicates that the call-leg is persistent. Otherwise, RV_FALSE.

**RETURN VALUES**
Returns RvStatus.
Call-leg Get and Set Functions
RvSipCallLegGetPersistency()

**RvSipCallLegGetPersistency()**

**DESCRIPTION**
Returns the *call-leg* persistency definition.

**SYNTAX**

```c
RvStatus RvSipCallLegGetPersistency(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvBool *pbIsPersistent);
```

**PARAMETERS**

- **hCallLeg**
The *call-leg* handle.

- **pbIsPersistent**
The *call-leg* persistency definition. RV_TRUE indicates that the *call-leg* is persistent. Otherwise, RV_FALSE.

**RETURN VALUES**
Returns RvStatus.
RvSipCallLegSetConnection()

DESCRIPTION

Sets a connection to be used by the call-leg transactions. The call-leg will hold this connection in its internal database. Whenever the call-leg creates a new transaction, it supplies the transaction with the given connection. The transaction will use the connection only if it fits the local and remote addresses of the transaction. Otherwise, the transaction will either locate a suitable connection in the connection hash or create a new connection. The call-leg will be informed that the supplied connection did not fit, and that a different connection was used and will update its database.

Note  This function can be used only if the call-leg is persistent.

SYNTAX

RvStatus RvSipCallLegSetConnection(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTransportConnectionHandle hConn);

PARAMETERS

hCallLeg
The call-leg handle.

hConn
The handle to the connection.

RETURN VALUES

Returns RvStatus.
RvSipCallLegGetConnection()

**DESCRIPTION**

Returns the connection that the call-leg transactions are currently using.

**Note**  Only persistent call-legs keep the currently-used connection. NULL will be returned if the call-leg is not persistent.

**SYNTAX**

RvStatus RvSipCallLegGetConnection(  
    IN RvSipCallLegHandle hCallLeg,  
    OUT RvSipTransportConnectionHandle *phConn);

**PARAMETERS**

- **hCallLeg**
  The handle to the call-leg.

- **phConn**
  The handle to the currently used connection.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegGetStackInstance()

**DESCRIPTION**

Returns the handle to the SIP Stack instance to which this *call-leg* belongs.

**SYNTAX**

```c
RvStatus RvSipCallLegGetStackInstance(
    IN RvSipCallLegHandle hCallLeg,
    OUT void* *phStackInstance);
```

**PARAMETERS**

- **hCallLeg**
  
  The handle to the *call-leg*.

- **phStackInstance**
  
  A valid pointer that will be updated with a handle to the SIP Stack instance.

**RETURN VALUES**

Returns *RvStatus*. 

Call-leg Get and Set Functions
RvSipCallLegSetRejectStatusCodeOnCreation()

RvSipCallLegSetRejectStatusCodeOnCreation()

DESCRIPTION
This function can be used synchronously from the RvSipCallLegCreatedEv() callback to instruct the Stack to automatically reject the request that created this call-leg. In this function you should supply the reject status code. If you set this status code, the call-leg will be destructed automatically when the RvSipCallLegCreatedEv() returns. The application will not get any further callbacks that relate to this call-leg. The application will not get the RvSipCallLegMsgToSendEv() for the reject response message or the TERMINATED state for the call-leg.

This function should not be used for rejecting a request in a normal scenario. For this you should use the RvSipCallLegReject() function. You should use this function only if your application is incapable of handling this new call-leg at all, for example, in a situation where an application is out of resources.

SYNTAX
RvStatus RvSipCallLegSetRejectStatusCodeOnCreation(
    IN RvSipCallLegHandle hCallLeg,
    IN RvUint16 rejectStatusCode);

PARAMETERS

hCallLeg
The handle to the call-leg.

rejectStatusCode
The reject status code for rejecting the request that created this object. The value must be from 300 to 699.

Note  When this function is used to reject a request, the application cannot use the outbound message mechanism to add information to the outgoing response message. If you wish to change the response message, you must use the regular reject function in the OFFERING state.
Call-leg Functions
RvSipCallLegSetRejectStatusCodeOnCreation()

**RETURN VALUES**

Returns RvStatus.
Call-leg Get and Set Functions

RvSipCallLegUseFirstRouteForInitialRequest()

**RvSipCallLegUseFirstRouteForInitialRequest()**

**DESCRIPTION**

An application may want to use a pre-loaded Route header when sending an INITIAL request. For this purpose, the application should add the Route header to the outbound message, and call this function to notify the Stack that it should send the request to the address of the first Route header in the outbound message.

**SYNTAX**

```c
RvStatus RvSipCallLegUseFirstRouteForInitialRequest(
    IN RvSipCallLegHandle hCallLeg);
```

**PARAMETERS**

- **hCallLeg**
  
  The handle to the *call-leg*.

**RETURN VALUES**

Returns **RvStatus**.
RvSipCallLegSetForceOutboundAddrFlag()

DESCRIPTION
Sets the “ForceOutboundAddr” flag. This flag forces the call-leg to send every request to the outbound address regardless of the message content or object state.

SYNTAX
RvStatus RvSipCallLegSetForceOutboundAddrFlag(
    IN RvSipCallLegHandle hCallLeg,
    IN RvBool bForceOutboundAddr);

PARAMETERS
hCallLeg
The call-leg handle.

bForceOutboundAddr
The flag value to set.

RETURN VALUES
Returns RvStatus.
Call-leg Get and Set Functions
RvSipCallLegSetAddAuthInfoToMsgFlag()

**RvSipCallLegSetAddAuthInfoToMsgFlag()**

**DESCRIPTION**
Sets the “AddAuthInfoToMsg” flag. If this flag is set to RV_FALSE, outgoing messages will not contain authorization information. The default value of this flag is RV_TRUE.

**SYNTAX**

```
RvStatus RvSipCallLegSetAddAuthInfoToMsgFlag(
    IN RvSipCallLegHandle hCallLeg,
    IN RvBool bAddAuthInfoToMsg);
```

**PARAMETERS**

- **hCallLeg**
  The call-leg handle.

- **bAddAuthInfoToMsg**
  The flag value to set.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegSetTranscTimers()

DESCRIPTION
Sets timeout values for the transaction timers of the call-leg. If some of the fields in pTimers are not set (UNDEFINED), this function will calculate them, or take the values from configuration.

SYNTAX
RvStatus RvSipCallLegSetTranscTimers(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTimers *pTimers,
    IN RvInt32 sizeOfTimersStruct);

PARAMETERS
hCallLeg
The handle to the call-leg.

pTimers
A pointer to the structure that contains all the timeout values.

sizeOfTimersStruct
The size of the RvSipTimers structure.

RETURN VALUES
Returns RvStatus.
**CALL-LEG HIGH AVAILABILITY FUNCTIONS**

The Call-leg High Availability API enables the application to store information on connected calls and restore the stored connected calls.

The Call-leg High Availability functions are:

- `RvSipCallLegGetConnectedCallStorageSize()`
- `RvSipCallLegStoreConnectedCall()`
- `RvSipCallLegRestoreConnectedCall()`
- `RvSipCallLegRestoreOldVersionConnectedCall()`
RvSipCallLegGetConnectedCallStorageSize()

DESCRIPTION

Gets the size of buffer needed to store all parameters of a connected call. (The size of buffer should be supplied in the function, RvSipCallLegStoreConnectedCall().

SYNTAX

RvStatus RvSipCallLegGetConnectedCallStorageSize(
    IN  RvSipCallLegHandle hCallLeg,
    OUT RvInt32 * len);

PARAMETERS

hCallLeg

The handle to the call-leg.

len

The size of buffer, which is -1 in case of failure.

RETURN VALUES

Returns RvStatus.
RvSipCallLegStoreConnectedCall()

DESCRIPTION

Copies all call-leg parameters from a given call-leg to a given buffer. This buffer should be supplied when restoring the call-leg. To store call-leg information, the call-leg must be in the CONNECTED state.

SYNTAX

RvStatus RvSipCallLegStoreConnectedCall(
    IN RvSipCallLegHandle hCallLeg,
    IN void* memBuff,
    IN RvUint32 buffLen);

PARAMETERS

hCallLeg

The handle to the call-leg.

memBuff

The buffer that will be filled with the call-leg information.

bufferLen

The length of the given buffer.

RETURN VALUES

Returns RvStatus.
RvSipCallLegRestoreConnectedCall()

DESCRIPTION
Restores all call-leg information into a given call-leg. The call-leg will assume the CONNECTED state and all call-leg parameters will be initialized from the given buffer.

SYNTAX
RvStatus RvSipCallLegRestoreConnectedCall(
    IN RvSipCallLegHandle hCallLeg,
    IN void* memBuff,
    IN RvUint32 buffLen);

PARAMETERS
hCallLeg
The handle to the call-leg.

memBuff
The buffer that stores the call-leg information.

buffLen
The buffer size.

RETURN VALUES
Returns RvStatus.
RvSipCallLegRestoreOldVersionConnectedCall()

**DESCRIPTION**
Restores *call-legs* that were stored by the SIP Stack version 3.0. This function restores *call-leg* information into a given *call-leg*. The *call-leg* will assume the CONNECTED state and all *call-leg* parameters will be initialized from the given buffer.

**SYNTAX**

RvStatus RvSipCallLegRestoreOldVersionConnectedCall(
    IN RvSipCallLegHandle hCallLeg,
    IN void* memBuff,
    IN RvUint32 buffLen,
    IN RvSipCallLegHARestoreMode eHAmode)

**PARAMETERS**

- **hCallLeg**
  The handle to the *call-leg*.

- **memBuff**
  The buffer that stores all the *call-leg* information.

- **buffLen**
  The buffer size.

- **eHAmode**
  Defines the exact mode used to store the given buffer.

**RETURN VALUES**

Returns RvStatus.
The Call-leg Server Authentication API enables the application to authenticate incoming Request messages.

The Call-leg Server Authentication functions are:

- RvSipCallLegAuthBegin()
- RvSipCallLegAuthProceed()
- RvSipCallLegRespondUnauthenticated()
- RvSipCallLegRespondUnauthenticatedDigest()
- RvSipCallLegGetCurrProcessedAuthObj()
- RvSipCallLegAuthObjGet()
- RvSipCallLegAuthObjRemove()
RvSipCallLegAuthBegin()  

**DESCRIPTION**  
Begins the server authentication process by challenging an incoming request. If the request is an active transaction (INVITE, BYE), hCallLeg should be given and hTransaction should be NULL. For a general request, hTransaction should be given.

**SYNTAX**  
RvStatus RvSipCallLegAuthBegin(  
    IN RvSipCallLegHandle hCallLeg,  
    IN RvSipTranscHandle hTransaction);  

**PARAMETERS**  

hCallLeg  
The handle to the call-leg the user wishes to challenge.

hTransaction  
The handle to the transaction the user wishes to challenge (in the case of a general request).

**RETURN VALUES**  
Returns RvStatus.
RvSipCallLegAuthProceed()

**DESCRIPTION**

Orders the SIP Stack to proceed with the authentication process. The information on how to proceed is given to the SIP Stack in the *action* parameter. The action options are:

- **RVSIP_TRANSC_AUTH_ACTION_USING_PASSWORD**
  Checks the given Authorization header with the given password.

- **RVSIP_TRANSC_AUTH_ACTION_SUCCESS**
  Orders to stop the authentication process with a success indication possibly because the user checked the Authorization header and it is correct. (This will cause AuthCompletedEv to be called, with Success status.)

- **RVSIP_TRANSC_AUTH_ACTION_FAILURE**
  The user wants to stop the loop that searches for Authorization headers. (This will cause AuthCompletedEv to be called, with Failure status.)

- **RVSIP_TRANSC_AUTH_ACTION_SKIP**
  Orders to skip the given header, and continue the authentication procedure with the next header, if it exists. (This will cause AuthCredentialFoundEv to be called, or AuthCompletedEv with Failure status, if there are no more Authorization headers.)

**SYNTAX**

```c
RvStatus RvSipCallLegAuthProceed(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransaction,
    IN RvSipTransactionAuthAction action,
    IN RvSipAuthorizationHeaderHandle hAuthorization,
    IN RvChar *password);
```

**PARAMETERS**

- **hCallLeg**
  The *call-leg* handle.
**Call-leg Server Authentication Functions**

**RvSipCallLegAuthProceed()**

- **hTransaction**
  The transaction handle.

- **action**
  Specifies which action to take to proceed with the authentication process.

- **hAuthorization**
  The handle to the Authorization header for which the function will check authentication. (This needed if the action is RVSIP_TRANSC_AUTH_ACTION_USING_PASSWORD, else NULL.)

- **password**
  The password for the realm+userName in the header. (This is needed if the action is RVSIP_TRANSC_AUTH_ACTION_USING_PASSWORD, else NULL.)

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegRespondUnauthenticated()

DESCRIPTION
Sends a 401/407 response. This function adds a copy of the given header to the
response message.

SYNTAX
RvStatus RvSipCallLegRespondUnauthenticated(
   IN RvSipCallLegHandle hCallLeg,
   IN RvSipTranscHandle hTransaction,
   IN RvUint16 responseCode,
   IN RvChar* strReasonPhrase,
   IN RvSipHeaderType headerType,
   IN void* hHeader);

PARAMETERS

hCallLeg
The call-leg handle.

hTransaction
The transaction handle.

responseCode
The response code: 401 or 407.

strReasonPhrase
The reason phrase for this response code. Can be NULL for the default reason
phrase.

headerType
The type of the given header.

hHeader
A pointer to the header to be set in the message.
Return Values

Returns RvStatus.
RvSipCallLegRespondUnauthenticatedDigest()

DESCRIPTION
Sends a 401/407 response. This function builds an Authentication header containing all the given parameters and adds it to the response message.

SYNTAX
RvStatus RvSipCallLegRespondUnauthenticatedDigest(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransaction,
    IN RvUint16 responseCode,
    IN RvChar *strReasonPhrase,
    IN RvChar *strRealm,
    IN RvChar *strDomain,
    IN RvChar *strNonce,
    IN RvChar *strOpaque,
    IN RvBool bStale,
    IN RvSipAuthAlgorithm eAlgorithm,
    IN RvChar *strAlgorithm,
    IN RvSipAuthQopOption eQop,
    IN RvChar *strQop);

PARAMETERS

hCallLeg
The callLeg handle.

hTransaction
The transaction handle.

responseCode
The response code: 401 for a user agent or 407 for proxies.

strReasonPhrase
The reason phrase for this response code.
Call-leg Server Authentication Functions
RvSipCallLegRespondUnauthenticatedDigest()

strRealm
The realm is mandatory.

strDomain
An optional string that may be NULL.

strNonce
An optional string that may be NULL.

strOpaque
An optional string that may be NULL.

bstale
RV_TRUE or RV_FALSE.

eAlgorithm
The enumeration of the algorithm. If RVSIP_AUTH_ALGORITHM_OTHER, the algorithm value is taken from the next argument.

strAlgorithm
The string of the algorithm. This parameter will be set only if the eAlgorithm parameter is set to be RVSIP_AUTH_ALGORITHM_OTHER.

eQop
The enumeration of Quality-of-Protection (Qop). If RVSIP_AUTH_QOP_OTHER, the Qop value will be taken from the next argument.

strQop
The string of Quality-of-Protection (Qop). This parameter will be set only if the eQop parameter is set to be RVSIP_AUTH_QOP_OTHER.

**RETURN VALUES**
Returns RvStatus.
RvSipCallLegGetCurrProcessedAuthObj()

**DESCRIPTION**

Retrieves the *authentication-object* that is currently being processed by the authenticator (for application usage in the RvSipAuthenticatorGetSharedSecretEv() callback).

**SYNTAX**

```c
RvStatus RvSipCallLegGetCurrProcessedAuthObj (  
    IN RvSipCallLegHandle hCallLeg,  
    OUT RvSipAuthObjHandle* phAuthObj);
```

**PARAMETERS**

- **hCallLeg**
  The handle to the *call-leg*.

- **phAuthObj**
  The *authentication-object* handle.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegAuthObjGet()

DESCRIPTION
Retrieves authentication-objects from the list in the call-leg. You can get the first, last, or next object. If you get the next object, you must supply the current object in the relative parameter.

SYNTAX
RvStatus RvSipCallLegAuthObjGet (  
    IN RvSipCallLegHandle hCallLeg,  
    IN RvSipListLocation eLocation,  
    IN RvSipAuthObjHandle hRelativeAuthObj,  
    OUT RvSipAuthObjHandle* phAuthObj);  

PARAMETERS

hCallLeg
The handle to the call-leg.

eLocation
The location in the list (first, last, or next).

hRelativeAuthObj
The relative object in the list (relevant for the next location).

phAuthObj
The authentication-object handle.

RETURN VALUES
Returns RvStatus.
RvSipCallLegAuthObjRemove()

DESCRIPTION

Removes an authentication-object from the list in the call-leg.

SYNTAX

RvStatus RvSipCallLegAuthObjRemove (  
    IN RvSipCallLegHandle    hCallLeg,  
    IN RvSipAuthObjHandle   hAuthObj);  

PARAMETERS

hCallLeg

The handle to the call-leg.

hAuthObj

The authentication-object handle.

RETURN VALUES

Returns RvStatus.
Call-leg Replaces Functions

**CALL-LEG REPLACES FUNCTIONS**

The Call-leg Replaces API enables the application to implement the Replaces draft.

The Call-leg Replaces functions are:

- `RvSipCallLegReplacesGetMatchedCallExt()`
- `RvSipCallLegReplacesCompareReplacesToCallLeg()`
RvSipCallLegReplacesGetMatchedCallExt()

DESCRIPTION
Called when the call-leg is in the OFFERING state. This function searches for the call-leg that has the same call-ID, to-tag and from-tag as the Replaces header in the original call-leg. If a matched call-leg is found, this call-leg is returned as the function output. Otherwise, this pointer will be NULL.

SYNTAX
RvStatus RvSipCallLegReplacesGetMatchedCallExt(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipCallLegReplacesReason *peReason,
    OUT RvSipCallLegHandle *hMatchedCallLeg);

PARAMETERS
hCallLeg
The handle to the call-leg that received the INVITE with the Replaces header.

peReason
If a dialog was found with same dialog identifiers but still does not match the Replaces header, this parameter indicates why the dialog does not fit. The application should use this parameter to decide how to respond (401/481/486/501) to the INVITE with the Replaces header.

hMatchedCallLeg
The handle to the call-leg matched the Replaces header. If there is no such call-leg, this handle will be NULL.

RETURN VALUES
Returns RvStatus.
**RvSipCallLegReplacesCompareReplacesToCallLeg()**

**DESCRIPTION**
A utility function that compares a call-leg to a Replaces header. The call-leg and Replaces header are equal if the Call-ID, from-tag, and to-tag are equal.

**SYNTAX**
```
RvStatus RvSipCallLegReplacesCompareReplacesToCallLeg(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipReplacesHeaderHandle hReplacesHeader,
    OUT RvBool *pbIsEqual);
```

**PARAMETERS**
- **hCallLeg**
The handle to the call-leg with which to compare the Replaces header.

- **hReplacesHeader**
The handle to a Replaces header.

- **pbIsEqual**
The result of the comparison. RV_TRUE if the call-leg and Replaces header are equal. Otherwise, RV_FALSE.

**RETURN VALUES**
Returns RvStatus.
The Call-leg Session-Timer API enables the application to get and set session-timer parameters, such as refresher type, Session-Expires, and min-SE in order to activate the session-timer functionality.

The Call-leg Session-Timer functions are:

- `RvSipCallLegSessionTimerInviteRefresh()`
- `RvSipCallLegTranscSessionTimerGeneralRefresh()`
- `RvSipCallLegSessionTimerSetPreferenceParams()`
- `RvSipCallLegTranscSessionTimerSetPreferenceParams()`
- `RvSipCallLegSessionTimerGetNegotiationParams()`
- `RvSipCallLegTranscSessionTimerGetNegotiationParams()`
- `RvSipCallLegSessionTimerGetAlertTime()`
- `RvSipCallLegSessionTimerSetAlertTime()`
- `RvSipCallLegSessionTimerGetRefresherType()`
- `RvSipCallLegSessionTimerGetMinSEValue()`
- `RvSipCallLegSessionTimerGetSessionExpiresValue()`
Call-leg Session-Timer Functions
RvSipCallLegSessionTimerInviteRefresh()

RvSipCallLegSessionTimerInviteRefresh()

DESCRIPTION

**Note**  Within a *call-leg* that handles a Session-Timer mechanism, this function has to be used instead of `RvSipCallLegRelInviteRequest()`. Otherwise, the Session-Timer parameter will be omitted from the re-INVITE request and the Session-Timer parameter will be turned off.

Sends a re-INVITE to refresh the session time. The refreshing re-INVITE request may be called in the following two situations:

- The *call-leg* already sent/received the 2xx response for the initial INVITE request (the *call-leg* state is CONNECTED or ACCEPTED or REMOTE-ACCEPTED).
- There are no other pending re-INVITE transactions (there is no other re-Invite object that is waiting for a final response).

SYNTAX

```c
RvStatus RvSipCallLegSessionTimerInviteRefresh(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipCallLegInviteHandle hInvite);
```

PARAMETERS

**hCallLeg**

The handle to the *call-leg* that the user wishes to refresh.

**hInvite**

The handle to the re-Invite object.

RETURN VALUES

Returns `RvStatus`.
RvSipCallLegTranscSessionTimerGeneralRefresh()

**DESCRIPTION**

Creates a transaction related to the call-leg and sends a Request message with the given method to refresh the call. “UPDATE” is the only general transaction that is allowed. The request will have the To, From, and Call-ID of the call-leg, and will be sent with a correct CSeq step. The request will be record-routed, if needed, and will include session-timer information.

**SYNTAX**

```c
RvStatus RvSipCallLegTranscSessionTimerGeneralRefresh(
    IN RvSipCallLegHandle hCallLeg,
    IN RvChar *strMethod,
    IN RvInt32 sessionExpires,
    IN RvInt32 minSE,
    IN RvSipCallLegSessionTimerRefresherPreference eRefresher,
    OUT RvSipTranscHandle *hTransc);
```

**PARAMETERS**

- **pCallLeg**
  A pointer to the call-leg the user wishes to modify.

- **strMethod**
  A string with the request method.

- **sessionExpires**
  The session time that will attach to this call.

- **minSE**
  The minimum Session-Expires time of this call.

- **eRefresher**
  The refresher preference for this call.
Call-leg Session-Timer Functions
RvSipCallLegTranscSessionTimerGeneralRefresh()

**hTransc**

The handle to the newly created *transaction*. If a transaction was supplied, this transaction will be used, and a new one will not be created.

**RETURN VALUES**

Returns *RvStatus*. 
Call-leg Session-Timer Functions
RvSipCallLegSessionTimerSetPreferenceParams()

**DESCRIPTION**
Sets the preference SessionTimer parameters associated with this call. These parameters may not be equal to the SessionTimer parameters of the call at the end of the negotiation (after a 200 OK has been sent or received).

**SYNTAX**
```
RvStatus RvSipCallLegSessionTimerSetPreferenceParams(
    IN RvSipCallLegHandle hCallLeg,
    IN RvInt32 sessionExpires,
    IN RvInt32 minSE,
    IN RvSipCallLegSessionTimerRefresherPreference eRefresher);
```

**PARAMETERS**

- **hCallLeg**
The handle to the *call-leg*.

- **sessionExpires**
The session time that will attach to this call.

- **minSE**
The minimum Session-Expires time of this call.

- **eRefresher**
The refresher preference for this call.

**RETURN VALUES**
Returns *RvStatus*.
REMARKS

The preference and negotiation parameters are temporary parameters used before the final session-timers parameters were determined. The preference parameters are the parameters that the local party prefers for this session. You can only set the preference parameters. The negotiation parameters are the parameters that the remote party prefers for this session. You can only get the negotiation parameters.

Before a UA sends a refresh request or response to a refresh request, the UA can set the preference parameters using the `RvSipCallLegSessionTimerSetPreferenceParams()` function. Upon receiving a refresh request or a 2xx response to a refresh request, the UA can learn about the remote party requested parameters using the `RvSipCallLegSessionTimerGetNegotiationParams()` function.

If the `sessionExpires` parameter is set to 0, the Session-Timer mechanism is turned off immediately in the current call-leg. Moreover, the Session-Timer mechanism can be turned on by:

- Calling this function with non-zero `sessionExpires` value in the middle of a call.
- Calling one of the Session-Timer Call-leg API functions for refreshing the current session (`RvSipCallLegTranscSessionTimerGeneralRefresh()` or `RvSipCallLegSessionTimerInviteRefresh()`).
RvSipCallLegTranscSessionTimerSetPreferenceParams()

DESCRIPTION

Sets the preference SessionTimer parameters associated with this transaction. "UPDATE" is the only general transaction that is allowed. These parameters may not be equal to the SessionTimer parameters of the call at the end of the negotiation (after 200 OK has been sent or received).

SYNTAX

RvStatus RvSipCallLegTranscSessionTimerSetPreferenceParams(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransc,
    IN RvInt32 sessionExpires,
    IN RvInt32 minSE,
    IN RvSipCallLegSessionTimerRefresherPreference eRefresher);

PARAMETERS

hCallLeg
The handle to the call-leg.

hTransc
The handle to the transaction to which the request belongs.

sessionExpires
The session time that will attach to this call.

minSE
The minimum Session-Expires time of this call.

eRefresher
The refresher preference for this call.
Call-leg Session-Timer Functions
RvSipCallLegTranscSessionTimerSetPreferenceParams()

RETURN VALUES
Returns RvStatus.

REMARKS
The preference and negotiation parameters are temporary parameters used before the final session-timers parameters were determined. The preference parameters are the parameters that the local party prefers for this session. You can only set the preference parameters. The negotiation parameters are the parameters that the remote party prefers for this session. You can only get the negotiation parameters.

Before a UA sends a refresh request or response to a refresh request, the UA can set the preference parameters using the 
RvSipCallLegSessionTimerSetPreferenceParams() function. Upon receiving a refresh request or a 2xx response to a refresh request, the UA can learn about the remote party requested parameters using the 
RvSipCallLegSessionTimerGetNegotiationParams() function.

If the sessionExpires parameter is set to 0, the Session-Timer mechanism is turned off immediately in the current call-leg. Moreover, the Session-Timer mechanism can be turned on by:

- Calling this function with non-zero sessionExpires value in the middle of a call.
- Calling one of the Session-Timer Call-leg API functions for refreshing the current session (RvSipCallLegTranscSessionTimerGeneralRefresh() or RvSipCallLegSessionTimerInviteRefresh()).
Call-leg Session-Timer Functions
RvSipCallLegSessionTimerGetNegotiationParams()

RvSipCallLegSessionTimerGetNegotiationParams()

DESCRIPTION

Gets the negotiation session-timer parameters associated with this call. These parameters may not be equal to the session-timer parameters of the call at the end of the negotiation (after a 200 OK has been sent or received).

SYNTAX

RvStatus RvSipCallLegSessionTimerGetNegotiationParams(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvInt32 *sessionExpires,
    OUT RvInt32 *minSE,
    OUT RvSipSessionExpiresRefresherType *eRefresherType,
    OUT RvSipCallLegSessionTimerRefresherPreference *eRefresherPref);

PARAMETERS

hCallLeg
The handle to the call-leg.

sessionExpires
The session time that will attach to this call.

minSE
The minimum Session-Expires time of this call.

eRefresherType
The refresher type for this call.

Return Values

Returns RvStatus.
Call-leg Session-Timer Functions
RvSipCallLegSessionTimerGetNegotiationParams()

REMARKS

The preference and negotiation parameters are temporary parameters used before the final session-timers parameters were determined. The preference parameters are the parameters that the local party prefers for this session. You can only set the preference parameters. The negotiation parameters are the parameters that the remote party prefers for this session. You can only get the negotiation parameters.

Before a UA sends a refresh request or response to a refresh request, the UA can set the preference parameters using the RvSipCallLegSessionTimerSetPreferenceParams() function. Upon receiving a refresh request or a 2xx response to a refresh request, the UA can learn about the remote party requested parameters using the RvSipCallLegSessionTimerGetNegotiationParams() function.
RvSipCallLegTranscSessionTimerGetNegotiationParams()

DESCRIPTION

Gets the negotiation session-timer parameters associated with this transaction. These parameters may not be equal to the session-timer parameters of the call at the end of the negotiation (after a 200 OK has been sent or received).

SYNTAX

RvStatus RvSipCallLegTranscSessionTimerGetNegotiationParams(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransc,
    OUT RvInt32 *sessionExpires,
    OUT RvInt32 *minSE,
    OUT RvSipSessionExpiresRefresherType *eRefresherType,
    OUT RvSipCallLegSessionTimerRefresherPreference *eRefresherPref);

PARAMETERS

hCallLeg

The handle to the call-leg.

hTransc

The handle to the transaction to which the request belongs.

sessionExpires

The session time that will attach to this call.

minSE

The minimum Session-Expires time of this call.

eRefresherType

The refresher type for this call.
Call-leg Session-Timer Functions
RvSipCallLegTranscSessionTimerGetNegotiationParams()

eRefresherPref
The refresher request for this call.

RETURN VALUES
Returns RvStatus.

REMARKS
The preference and negotiation parameters are temporary parameters used before the final session-timers parameters were determined. The preference parameters are the parameters that the local party prefers for this session. You can only set the preference parameters. The negotiation parameters are the parameters that the remote party prefers for this session. You can only get the negotiation parameters.

Before a UA sends a refresh request or response to a refresh request, the UA can set the preference parameters using the RvSipCallLegSessionTimerSetPreferenceParams() function. Upon receiving a refresh request or a 2xx response to a refresh request, the UA can learn about the remote party requested parameters using the RvSipCallLegSessionTimerGetNegotiationParams() function.
RvSipCallLegSessionTimerGetAlertTime()  

DESCRIPTION  
Returns the time in which the Refresh will be sent after the session starts in 
automode, or calls the RvSipCallLegSessionTimerRefreshAlertEv() callback in 
manual mode.  

SYNTAX  
RvStatus RvSipCallLegSessionTimerGetAlertTime(
  IN RvSipCallLegHandle hCallLeg,
  OUT RvInt32 *alertTime);

PARAMETERS  

hCallLeg  
The handle to the call-leg.  

alertTime  
The time in which the Refresh will be send before the session ends.

RETURN VALUES  
Returns RvStatus.
Call-leg Session-Timer Functions
RvSipCallLegSessionTimerSetAlertTime()

RvSipCallLegSessionTimerSetAlertTime()

DESCRIPTION
Enables the application to modify the time in which the refresh will be sent after
the session starts in auto mode, or calls the
RvSipCallLegSessionTimerRefreshAlertEv() callback in manual mode. (The
default time is sessionExpires/2).

SYNTAX
RvStatus RvSipCallLegSessionTimerSetAlertTime(
    IN RvSipCallLegHandle hCallLeg,
    IN RvInt32 alertTime);

PARAMETERS
hCallLeg
The handle to the call-leg.

alertTime
The time in which a Refresh will be send before the session ends.

RETURN VALUES
Returns RvStatus.
RvSipCallLegSessionTimerGetRefresherType()

**DESCRIPTION**

Returns the refresher type of this call. The value of the refresher type can be different from the refresher preference (which is the request of the application).

**SYNTAX**

```c
RvStatus RvSipCallLegSessionTimerGetRefresherType(
    IN RvSipCallLegHandle hCallLeg, 
    OUT RvSipSessionExpiresRefresherType *eRefresher);
```

**PARAMETERS**

- **hCallLeg**
  
The handle to the call-leg.

- **eRefresher**
  
The refresher request for this call.

**RETURN VALUES**

Returns RvStatus.
### RvSipCallLegSessionTimerGetMinSEValue()

**DESCRIPTION**

Returns the Min-SE value associated with this call.

**SYNTAX**

```c
RvStatus RvSipCallLegSessionTimerGetMinSEValue(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvInt32 *minSE);
```

**PARAMETERS**

- **hCallLeg**
  
  The handle to the *call-leg*.

- **minSE**
  
  The minimum Session-Expires time of this call.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegSessionTimerGetSessionExpiresValue()

DESCRIPTION
Returns the Session-Expires value associated with this call.

SYNTAX
RvStatus RvSipCallLegSessionTimerGetSessionExpiresValue(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvInt32 *sessionExpires);

PARAMETERS
hCallLeg
The handle to the call-leg.

sessionExpires
The Session-Expires time of this call.

RETURN VALUES
Returns RvStatus.
CALL-LEG DNS FUNCTIONS

The Call-leg DNS API enables the application to work with the advanced DNS feature in the Call-leg layer. When sending a request failed due to a network error, timeout, or receiving a 503 response, the application can re-send the Request message to the next address in the transaction DNS list or give up sending the request.

The Call-leg DNS functions are:

- RvSipCallLegDNSGiveUp()
- RvSipCallLegDNSContinue()
- RvSipCallLegDNSReSendRequest()
- RvSipCallLegDNSGetList()
RvSipCallLegDNSGiveUp()

DESCRIPTION

Stops retrying to send a request after send failure. When a call-leg fails to send a request due to a timeout, network error, or 503 response, the call-leg moves to the MSG_SEND_FAILURE state. In this state, the application can do one of the following:

- Send the request to the next DNS-resolved IP.
- Give up on this request.

Calling RvSipCallLegDNSGiveUp() indicates that the application wishes to give up on this request. Retrying to send the request will stop and the call-leg will change its state back to the previous state. You can use this function for a call-leg request or for a general request, as follows:

- Use this function for INVITE, BYE and re-INVITE messages, if the state was changed to MSG_SEND_FAILURE state. For these methods, you should set the transaction handle parameter to NULL.
- Use this function for a general request if you received a MSG_SEND_FAILURE status in the RvSipCallLegTranscStateChangedEv() event. For such a general request, you should supply the general transaction handle.

SYNTAX

RvStatus RvSipCallLegDNSGiveUp(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransc);

PARAMETERS

hCallLeg

The handle to the call-leg that sent the request.

hTransc

The handle to the transaction, in the case of a general request.
Call-leg DNS Functions
RvSipCallLegDNSGiveUp()

RETURN VALUES

Returns RvStatus.
RvSipCallLegDNSContinue()

DESCRIPTION
Prepares the call-leg to retry sending a request after the previous attempt failed. When a call-leg fails to send a request due to a timeout, network error, or 503 response, the call-leg moves to the MSG_SEND_FAILURE state. In this state, the application can do one of the following:

- Send the request to the next DNS-resolved IP.
- Give up on this request.

To retry sending the message, the application must first call the RvSipCallLegDNSContinue() function. Calling this function closes the failure transaction and updates the DNS list. (To re-send the request to the next IP address, use the RvSipCallLegDNSReSendRequest() function).

You can use this function for a call-leg request or for a general request, as follows:

- Use this function for Invite, BYE and re-INVITE messages, if the state was changed to MSG_SEND_FAILURE state. For these methods, you should set the transaction handle parameter to NULL.
- Use this function for a general request if you received a MSG_SEND_FAILURE status in the RvSipCallLegTranscStateChangedEv() event. For such a general request, you should supply the general transaction handle.

Calling this function for a failure resulting from a 503 response will send an ACK on the INVITE.

SYNTAX

RvStatus RvSipCallLegDNSContinue(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransc,
    OUT RvSipTranscHandle* phClonedTransc);

Call-leg DNS Functions
RvSipCallLegDNSContinue()
Call-leg DNS Functions
RvSipCallLegDNSContinue()

PARAMETERS

hCallLeg
The handle to the call-leg that sent the request.

hTransc
The handle to the transaction, in the case of a general request.

phClonedTransc
The handle to the cloned transaction, in the case of a general transaction only.

RETURN VALUES
Returns RvStatus.
RvSipCallLegDNSReSendRequest()

**DESCRIPTION**

Re-sends a request after the previous attempt failed. When a call-leg fails to send a request due to a timeout, network error, or 503 response, the call-leg object moves to the MSG_SEND_FAILURE state. In this state, the application can do one of the following:

- Send the request to the next DNS resolved IP.
- Give up on this request.

To re-send the Register request, first call RvSipCallLegDNSContinue(). You should then call RvSipCallLegDNSReSendRequest(). The request will automatically be sent to the next resolved IP address in the DNS list. You can use this function for a call-leg request or for a general request, as follows:

- Use this function for Invite, BYE and re-INVITE messages, if the state was changed to MSG_SEND_FAILURE state. For these methods, you should set the transaction handle parameter to NULL.
- Use this function for a general request if you received a MSG_SEND_FAILURE status in the RvSipCallLegTranscStateChangedEv() event. For such a general request, you should supply the general transaction handle.

**SYNTAX**

```c
RvStatus RvSipCallLegDNSReSendRequest(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransc);
```

**PARAMETERS**

- **hCallLeg**
  The handle to the call-leg that sent the request.

- **hTransc**
  The handle to the transaction, in the case of a general request.
Call-leg DNS Functions
RvSipCallLegDNSReSendRequest()

**RETURN VALUES**

Returns `RvStatus`. 
RvSipCallLegDNSGetList()

DESCRIPTION
Retrieves a DNS list object from the call-leg current active or requested transaction. When a call-leg fails to send a request due to a timeout, network error, or 503 response, the call-leg moves to the MSG_SEND_FAILURE state. In this state you can use RvSipCallLegDNSGetList() to get the DNS list if you wish to view or change it. You can use this function for a call-leg request or for a general request, as follows:

- Use this function for Invite, BYE and re-INVITE messages, if the state was changed to MSG_SEND_FAILURE state. For these methods, you should set the transaction handle parameter to NULL.
- Use this function for a general request if you received a MSG_SEND_FAILURE status in the RvSipCallLegTranscStateChangedEv() event. For such a general request, you should supply the general transaction handle.

SYNTAX
RvStatus RvSipCallLegDNSGetList(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTranscHandle hTransc,
    OUT RvSipTransportDNSListHandle *phDnsList);

PARAMETERS

hCallLeg
The handle to the call-leg that sent the request.

hTransc
The handle to the transaction in the case of a general request.

phDnsList
The DNS list handle.
RETURN VALUES

Returns RvStatus.
The Call-leg Local and Remote Address functions enable the application to control the addresses that the call-leg uses for sending requests. The Call-leg Local and Remote Address functions are:

- `RvSipCallLegSetLocalAddress()`
- `RvSipCallLegGetLocalAddress()`
- `RvSipCallLegSetOutboundDetails()`
- `RvSipCallLegGetOutboundDetails()`
- `RvSipCallLegGetTranscCurrentLocalAddress()`
RvSipCallLegSetLocalAddress()

DESCRIPTION

Sets the local address from which the call-leg will send outgoing requests. The SIP Stack 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/IPv6). When the SIP Stack 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 (transport type and address type). If several configured local addresses match the remote address characteristics, the first configured address is taken. You can use RvSipCallLegSetLocalAddress() to force the call-leg to choose a specific local address for a specific transport and address type. For example, you can force the call-leg to use the second configured UDP/IPv4 local address. If the call-leg 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 in the initialization of the SIP Stack. If you configured the SIP Stack to listen to a 0.0.0.0 local address, you must use the same notation here.

SYNTAX

RvStatus RvSipCallLegSetLocalAddress(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTransport eTransportType,
    IN RvSipTransportAddressType eAddressType,
    IN RvChar *localAddress,
    IN RvUint16 localPort);

PARAMETERS

hCallLeg

The handle to the call-leg.
**Call-leg Local and Remote Address Functions**

RvSipCallLegSetLocalAddress()

- **eTransportType**
  The transport type (UDP, TCP or TLS).

- **eAddressType**
  The address type (IPv4 or IPv6).

- **localAddress**
  The local address to be set to this call-leg.

- **localPort**
  The local port to be set to this call-leg.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegGetLocalAddress()

DESCRIPTION

Gets the local address which the call-leg will use to send outgoing requests to a destination that listens to a specific transport and address type. This is the address the user set using the RvSipCallLegSetLocalAddress() function. If no address was set, the function will return the default, first configured local address of the requested transport and address type.

SYNTAX

RvStatus RvSipCallLegGetLocalAddress(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTransport eTransportType,
    IN RvSIPTransportAddressType eAddressType,
    OUT RvChar *localAddress,
    OUT RvUint16 *localPort);

PARAMETERS

hCallLeg
The handle to the call-leg.

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

eAddressType
The address type (IP4 or IP6).

localAddress
The local address this call-leg is using.

localPort
The local port this call-leg is using.
Call-leg Local and Remote Address Functions
RvSipCallLegGetLocalAddress()

**RETURN VALUES**

Returns *RvStatus*. 
**RvSipCallLegSetOutboundDetails**

**DESCRIPTION**
Sets all outbound proxy details to the call-leg. All details are supplied in the RvSipTransportOutboundProxyCfg structure that includes parameters such as the IP address or host name transport and port type. Requests that this object sends 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 list.

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

**SYNTAX**

RvStatus RvSipCallLegSetOutboundDetails(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipTransportOutboundProxyCfg *pOutboundCfg,
    IN RvInt32 sizeOfCfg);

**PARAMETERS**

**hCallLeg**
The handle to the call-leg.

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

**sizeOfCfg**
The size of the outbound proxy configuration structure.

**RETURN VALUES**
Returns RvStatus.
Call-leg Local and Remote Address Functions

RvSipCallLegGetOutboundDetails()

**DESCRIPTION**

Gets all outbound proxy details that the call-leg uses. The details are placed in the RvSipTransportOutboundProxyCfg structure that includes parameters such as the IP address or host name transport and port type. If the outbound details were not set to the specific call-leg, rather the outbound proxy was defined to the SIP Stack upon initialization, these parameters will be returned. If the call-leg 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**

```c
RvStatus RvSipCallLegGetOutboundDetails(
    IN RvSipCallLegHandle hCallLeg,
    IN RvInt32 sizeOfCfg,
    OUT RvSipTransportOutboundProxyCfg *pOutboundCfg);
```

**PARAMETERS**

- **hCallLeg**
  The handle to the call-leg.

- **sizeOfCfg**
  The size of the configuration structure.

- **pOutboundCfg**
  A pointer to the outbound proxy configuration structure.

**RETURN VALUES**

Returns RvStatus.
RvSipCallLegGetTranscCurrentLocalAddress()

DESCRIPTION

Gets the local address that is used by a specific call-leg transaction. You can supply a specific transaction handle or use NULL to get the address of the active transaction (INVITE transaction). In this version, this function can be called only in the OFFERING state to get the local address of an incoming INVITE transaction.

SYNTAX

```c
RvStatus RvSipCallLegGetTranscCurrentLocalAddress(
    IN  RvSipCallLegHandle hCallLeg,
    IN  RvSipTranscHandle hTransc,
    OUT RvSipTransport *pTransportType,
    OUT RvSipTransportAddressType *pAddressType,
    OUT RvChar *localAddress,
    OUT RvUint16 *localPort);
```

PARAMETERS

**hCallLeg**

The handle to the call-leg.

**hTransc**

The handle to a specific call-leg transaction or NULL to indicate an active transaction.

**eTransportType**

The transport type (UDP, TCP or TLS).

**eAddressType**

The address type (IP4 or IP6).

**localAddress**

The local address that this transaction is using (must be longer than 48 bytes).
Call-leg Local and Remote Address Functions

RvSipCallLegGetTranscCurrentLocalAddress()

**localPort**

The local port that this transaction is using.

**RETURN VALUES**

Returns `RvStatus`.
CALL-LEG FORKING FUNCTIONS

The Call-leg Forking functions are:

- RvSipCallLegSetForkingEnabledFlag()
- RvSipCallLegGetForkingEnabledFlag()
- RvSipCallLegSetForked1xxTimerTimeout()
Call-leg Forking Functions
RvSipCallLegGetOriginalCallLeg()

RvSipCallLegGetOriginalCallLeg()

DESCRIPTION
An initial INVITE request may be forked by a proxy and as a result, several 1xx and 2xx responses may be received from several UASs. The first incoming response will be mapped to the original call-leg that sent the INVITE request. Every other incoming response (with a different to-tag parameter) creates a new “forked call-leg”.

This function returns the original call-leg of a given call-leg: If the given call-leg is a forked call-leg, the function will return the original call-leg handle. If the given call-leg is an original call-leg, the function will return the same call-leg handle.

SYNTAX
RvStatus RvSipCallLegGetOriginalCallLeg(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipCallLegHandle *phOrigCallLeg);

PARAMETERS

hCallLeg
The handle to a call-leg.

phOrigCallLeg
The handle to the original call-leg.

RETURN VALUES
Returns RvStatus.
RvSipCallLegSetForkingEnabledFlag()

DESCRIPTION

Sets the “forking enabled” flag of the call-leg. This flag defines the call-leg behavior upon receiving a forked 1xx/2xx response. If this flag is set to TRUE, a new call-leg is created for every 1xx/2xx response with a new to-tag. If this flag is set to FALSE, the response will be mapped to the original call-leg. Every 1xx response will update the to-tag parameter of the call-leg.

SYNTAX

RvStatus RvSipCallLegSetForkingEnabledFlag ( IN RvSipCallLegHandle hCallLeg, IN RvBool bForkingEnabled);

PARAMETERS

hCallLeg

The handle to the call-leg.

RETURN VALUES

Returns RvStatus.
RvSipCallLegGetForkingEnabledFlag()

DESCRIPTION

Returns the “forking enabled” flag of the call-leg. This flag defines the call-leg behavior on receiving a forked 1xx/2xx response. If this flag is set to TRUE, a new call-leg is created for every 1xx/2xx response with a new to-tag. If this flag is set to FALSE, the response will be mapped to the original call-leg. Every 1xx response will update the to-tag parameter of the call-leg.

SYNTAX

RvStatus RvSipCallLegGetForkingEnabledFlag(
    IN RvSipCallLegHandle hCallLeg,
    IN RvBool *pbForkingEnabled);

PARAMETERS

hCallLeg

The handle to the call-leg.

RETURN VALUES

Returns RvStatus.
Call-leg Forking Functions
RvSipCallLegSetForked1xxTimerTimeout()

RvSipCallLegSetForked1xxTimerTimeout()

DESCRIPTION
Sets the timeout value for the forked-1xx-timer. An initial INVITE request may be forked by a proxy, and as a result, several 1xx and 2xx responses may be received from several UASs. The first incoming response will be mapped to the original call-leg that sent the INVITE request. Every other incoming response (with a different to-tag parameter) creates a new “forked call-leg”. A forked call-leg that received a 1xx response sets a timer (forked-1xx-timer). This timer will be released when this call-leg receives a 2xx response. If it expires before the 2xx was received, the call-leg is terminated. (usually the timer timeout value is taken from the SIP Stack configuration).

SYNTAX
RvStatus RvSipCallLegSetForked1xxTimerTimeout (  
   IN RvSipCallLegHandle hCallLeg,  
   IN RvInt32 timeout);

PARAMETERS

hCallLeg
The handle to the call-leg.

timeout
The timeout value for the 1xx timer. (If 0, the timer will not be set.)

RETURN VALUES
Returns RvStatus.
The Transaction API functions of the SIP Stack enable you to create and manage Transactions according to RFC 3261.

This part includes the following section:

- Transaction Functions
The Transaction API functions of the SIP Stack enable you to create and manage transaction objects (transaction) according to RFC 3261. Using the API, you can implement SIP servers, such as Registrar, Redirect Server, or a Proxy. You can also create a transaction, send the request, and receive the response using callbacks. Incoming transactions that are not related to a call-leg are also handled by the Transaction layer. You will be informed of such transactions with callback functions. This section contains the Transaction Manager functions included in the RvSipTransaction.h header file.

The functions included in this section are:

- Transaction Manager Functions
- Transaction Control Functions
- Transaction Get and Set Functions
- Transaction Authentication Functions
- Transaction DNS Functions
- Transaction Local and Remote Address Functions
The Transaction Manager (TransactionMgr) is responsible for all the transactions. Transaction Manager API functions enable you to set the event handlers of the Transaction module and to create new transactions.

The Transaction Manager API functions are:

- RvSipTransactionMgrSetEvHandlers()
- RvSipTranScMgrCreateTransaction()
- RvSipTranScMgrCreateServerTransactionFromMsg()
- RvSipTransactionMgrSetMgrEvHandlers()
- RvSipTranScMgrSetAppErrMsgHandle()
- RvSipTranScMgrGetAppErrMsgHandle()
- RvSipTranScMgrGetStackInstance()
RvSipTransactionMgrSetEvHandlers()

DESCRIPTION
Sets event handlers for all transaction events.

SYNTAX
RvStatus RvSipTransactionMgrSetEvHandlers(
    IN RvSipTranscMgrHandle hTranscMgr,
    IN void *pAppContext,
    IN RvSipTransactionEvHandlers *pHandlers,
    IN RvUint32 evHandlerStructSize);

PARAMETERS

hTranscMgr
The handle to the TransactionMgr.

pAppContext
An application context. This context is supplied with the callback informing of the creation of a new transaction.

pHandlers
The pointer to the structure containing application event handler pointers.

evaHandlerStructSize
The size of the event handler structure.

RETURN VALUES
Returns RvStatus.
RvSipTranscMgrCreateTransaction()

DESCRIPTION
Creates a new client *transaction* and exchanges handles with the application. The new *transaction* assumes the IDLE state.

SYNTAX

```
RvStatus RvSipTranscMgrCreateTransaction(
    IN  RvSipTranscMgrHandle hTranscMgr,
    IN  RvSipTranscOwnerHandle hTranscOwner,
    OUT RvSipTranschHandle    *hTransc);
```

PARAMETERS

**hTranscMgr**

The handle to the *TransactionMgr*.

**hTranscOwner**

The owner of the *transaction*, which is the application handle for this *transaction*. The term “owner” is used since a *transaction* can also be owned by a *call-leg* or a *register client*, and not by the application.

**hTransc**

SIP Stack handle to the newly created *transaction*.

RETURN VALUES

Returns *RvStatus*. 
**RvSipTranscMgrCreateServerTransactionFromMsg()**

**DESCRIPTION**

Creates a new Server transaction from a Request message that the application supplies. This function creates a new transaction in the REQUEST_RECEIVED state.

**SYNTAX**

```c
RvStatus RvSipTranscMgrCreateServerTransactionFromMsg(
    IN RvSipTranscMgrHandle hTranscMgr,
    IN RvSipTranscOwnerHandle hTranscOwner,
    IN RvSipMsgHandle hMsg,
    OUT RvSipTranscHandle *hTransc);
```

**PARAMETERS**

- **hTranscMgr**
  The TransactionMgr to which this transaction will belong.

- **hTranscOwner**
  The application owner of the transaction.

- **hMsg**
  The handle to the Request message that the application supplies to create the transaction.

- **hTransc**
  The new transaction that was created. If the function failed, hTransc = NULL.

**RETURN VALUES**

Returns RvStatus.
Transaction Manager Functions
RvSipTransactionMgrSetMgrEvHandlers()

RvSipTransactionMgrSetMgrEvHandlers()

DESCRIPTION
Sets a set of event handlers that the TransactionMgr will use. TransactionMgr events do not belong to a specific transaction but to the TransactionMgr itself.

SYNTAX
RvStatus RvSipTransactionMgrSetMgrEvHandlers(
    IN RvSipTranscMgrHandle hTranscMgr,
    IN void *pAppContext,
    IN RvSipTransactionMgrEvHandlers *pMgrHandlers,
    IN RvUint32 evHandlerStructSize)

PARAMETERS
hTranscMgr
A handle to the TransactionMgr.

pAppContext
An application context. This context will be supplied when the TransactionMgr callbacks will be called.

pMgrHandlers
A pointer to a structure of event handlers.

evHandlerStructSize
The size of the pMgrHandlers structure.

RETURN VALUES
Returns RvStatus.
**RvSipTranscMgrSetAppMgrHandle()**

**DESCRIPTION**
Sets the application TransactionMgr handle in the SIP Stack TransactionMgr. The application can use this handle to save a context in the TransactionMgr. This handle is given in the transaction created callback.

**SYNTAX**

```c
RvStatus RvSipTranscMgrSetAppMgrHandle(
    IN RvSipTranscMgrHandle hTranscMgr,
    IN void* hAppMgr);
```

**PARAMETERS**

- **hTranscMgr**
  The SIP Stack TransactionMgr handle.

- **hAppMgr**
  The application TransactionMgr handle.

**RETURN VALUES**

Returns RvStatus.
RvSipTranscMgrGetAppMgrHandle()

DESCRIPTION
Gets the application TransactionMgr handle from the SIP Stack TransactionMgr.

SYNTAX
RvStatus RvSipTranscMgrGetAppMgrHandle(
    IN RvSipTranscMgrHandle hTranscMgr,
    OUT void** hAppMgr);

PARAMETERS
hTranscMgr
The SIP Stack TransactionMgr handle.

hAppMgr
The application TransactionMgr handle.

RETURN VALUES
Returns RvStatus.
RvSipTranscMgrGetStackInstance()

DESCRIPTION

Returns the handle to the Stack instance to which this TransactionMgr belongs.

SYNTAX

RvStatus RvSipTranscMgrGetStackInstance(
    IN RvSipTranscMgrHandle hTranscMgr,
    OUT void** phStackInstance);

PARAMETERS

hTranscMgr

The handle to the TransactionMgr.

phStackInstance

A valid pointer which will be updated with a handle to the Stack instance.

RETURN VALUES

Returns RvStatus.
The Transaction API functions enable you to create new transactions, send and receive requests and responses, and terminate transactions.

The Transaction Control Functions are:

- RvSipTransactionRequest()
- RvSipTransactionRespond()
- RvSipTransactionMake()
- RvSipTransactionCancel()
- RvSipTransactionAck()
- RvSipTransactionTerminate()
- RvSipTransactionRequestMsg()
- RvSipTransactionRespondMsg()
- RvSipTransactionIsUnsupportedExtRequired()
- RvSipTransactionIgnoreOutboundProxy()
- RvSipTransactionSendToFirstRoute()
- RvSipTransactionDetachOwner()
**RvSipTransactionRequest()**

**DESCRIPTION**

Sends a Request message. The method of the request and the request URI are given as parameters to this function. This function can be called only after the To and From fields have been set.

**Note**  If you want to add additional headers to the outgoing request, you should use the outbound message mechanism. For more information, see RvSipTransactionGetOutboundMsg().

**SYNTAX**

```c
RvStatus RvSipTransactionRequest(
    IN RvSipTranscHandle hTransc,
    IN RvChar *strRequestMethod,
    IN RvSipAddressHandle hRequestUri);
```

**Parameters**

- **hTransc**
  The *transaction* handle.

- **strRequestMethod**
  The request method as a string.

- **hRequestUri**
  The request URI to be used in the Request message. The request URI of the message is the address given here, and the message will be sent accordingly.

**RETURN VALUES**

Returns RvStatus.
**RvSipTransactionRespond()**

**DESCRIPTION**

Sends a response message with the given response code and a reason phrase. The reason phrase is optional and can be set to NULL. In this case, the response will have a default reason phrase according to RFC 3261. You can use this function for both provisional and final responses.

**Note**  If you want to add additional headers to the outgoing response, you should use the outbound message mechanism. For more information, see RvSipTransactionGetOutboundMsg().

**SYNTAX**

```c
RvStatus RvSipTransactionRespond(
    IN RvSipTranscHandle hTransc,
    IN RvUint16 responseCode,
    IN RvChar *strReasonPhrase);
```

**PARAMETERS**

- **hTransc**
  
  The `transaction` handle.

- **responseCode**
  
  The response code with which to respond.

- **strReasonPhrase**
  
  The reason phrase to describe the response code. If set to NULL, a default reason phrase will be supplied.

**RETURN VALUES**

Returns `RvStatus`. 
RvSipTransactionMake()

DESCRIPTION

Initializes a transaction and sends a request. The method of the request and the Request URI are given as parameters. Before calling this function you should call RvSipTranscMgrCreateTransaction() to create a new transaction. A Call-ID is created automatically by the Transaction layer. You can use RvSipTransactionSetCallId() before calling this function to specify a different Call-ID.

SYNTAX

RvStatus RvSipTransactionMake(
    IN RvSipTranscHandle hTransc,
    IN RvChar* strFrom,
    IN RvChar* strTo,
    IN RvChar* strRequestURI,
    IN RvInt32 cseqStep,
    IN RvChar* strMethod);

PARAMETERS

hTransc
The transaction handle.

strFrom
A string with the From party header. For example, “From:sip:172.20.1.1:5060”.

strTo
A string with the To party header. For example, “To:sip:172.20.5.5:5060”.

strRequestURI
A string with the request URI address. For example, “sip:sara@172.20.1.1:5060”.
Transaction Control Functions
RvSipTransactionMake()  

`cseqStep`

The CSeq step to use in this transaction.

`strMethod`

The request method as a string.

**RETURN VALUES**

Returns `RvStatus`.  


RvSipTransactionCancel()

DESCRIPTION
Cancels the transaction given as a parameter. Calling this function causes a client CANCEL transaction to be created with a transaction key taken from the transaction that is about to be cancelled. The CANCEL request is then sent to the remote party. If the transaction to cancel is an INVITE transaction, it will assume the INVITE_CANCELLED state. A general transaction will assume the general cancelled state. The newly created CANCEL transaction will assume the CANCEL_SENT state.

You can only cancel transactions that received a provisional response and are in the PROCEEDING or PROCEEDING_TIMEOUT states. Cancelling a transaction that has already received a final response has no effect.

SYNTAX
RvStatus RvSipTransactionCancel(
    IN RvSipTranscHandle hTransc);

PARAMETERS
hTransc
The transaction to cancel.

RETURN VALUES
Returns RvStatus.
**RvSipTransactionAck()**

**DESCRIPTION**

Sends an ACK request on a given INVITE transaction. This function can only be called on an INVITE client transaction in the RVSIP_TRANSC_STATE_CLIENT_INVITE_FINAL_RESPONSE_RCVD state when the response is a non-2xx response. According to RFC 3261, an ACK request for an INVITE transaction that received a 2xx response is not part of the transaction. This ACK should be sent using the Transmitter API.

**Note** When the “bOldInviteHandling” configuration parameter is set to RV_TRUE, the ACK request is part of the transaction even for 2xx responses. In this case, you will be able to use this function for sending the ACK request for all transaction response codes. This mode of action, however, is not recommended.

**SYNTAX**

```c
RvStatus RvSipTransactionAck(
    IN RvSipTranscHandle hTransc,
    IN RvSipAddressHandle hRequestUri);
```

**PARAMETERS**

- **hTransc**
  
  The transaction handle.

- **hRequestUri**
  
  The Request-URI to use in the ACK message. If the request URI is NULL, the request URI of the transaction is used as the request URI. The request URI can be NULL when the ACK is to a non 2xx response, and then the request URI must be the same as the INVITE request URI.

**RETURN VALUES**

Returns RvStatus.
RvSipTransactionTerminate()

DESCRIPTION
Causes immediate shut-down of the transaction. The transaction will assume the TERMINATED state.

SYNTAX
RvStatus RvSipTransactionTerminate(
    IN RvSipTranscHandle hTransc);

PARAMETERS
hTransc
The transaction to terminate.

RETURN VALUES
Returns RvStatus.
RvSipTransactionRequestMsg()

**DESCRIPTION**
Sends a Request message to the remote party using a given prepared message object.

**SYNTAX**
```c
RvStatus RvSipTransactionRequestMsg(
    IN RvSipTranscHandle hTransc,
    IN RvSipMsgHandle hMsg,
    IN RvBool bAddTopVia);
```

**PARAMETERS**

**hTransc**
The transaction handle.

**hMsg**
The handle to the Request message object. This message is sent to the remote party.

**bAddTopVia**
Indicates whether the SIP Stack 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 Stack to handle the Via header for you. If you have added the Via header by yourself, set this parameter to RV_FALSE.

**RETURN VALUES**
Returns RvStatus.
Remarks

- This function is normally used by proxy implementations in order to proxy a received request to the remote party. The request will be sent according to the RequestURI found in the message object.

- The user is responsible for setting the correct Request-URI in the given message object and for applying the Record-Route rules if necessary.
RvSipTransactionRespondMsg()

DESCRIPTION
Sends a Response message to the remote party using a given prepared message object.

SYNTAX
RvStatus RvSipTransactionRespondMsg(
    IN RvSipTranscHandle hTransc,
    IN RvSipMsgHandle hMsg,
    IN RvBool bRemoveTopVia);

PARAMETERS
hTransc
The transaction handle.

hMsg
The handle to Respond message object.

bRemoveTopVia
Indicates whether the SIP Stack should remove the top Via header from the message object before the destination is resolved and the message is sent.

RETURN VALUES
Returns RvStatus.

REMARKS
This function is normally used by proxy implementations in order to proxy a received response. The Respond message will be sent according to the top Via header found in the message object.
**RvSipTransactionIsUnsupportedExtRequired()**

**DESCRIPTION**

Returns whether or not the transaction requires an extension that is not supported. You can call this function in states that indicate that a request was received. You can use this function only if the rejectUnsupportedExtensions configuration parameter is set to RV_TRUE.

**SYNTAX**

```c
RvStatus RvSipTransactionIsUnsupportedExtRequired(
    IN RvSipTranscHandle hTransc,
    OUT RvBool *bUnsupportedReq);
```

**PARAMETERS**

- **hTransc**
  
  The transaction handle.

- **bUnsupportedReq**
  
  RV_TRUE if the request requires unsupported extension. RV_FALSE otherwise.

**RETURN VALUES**

Returns RvStatus.
**RvSipTransactionIgnoreOutboundProxy()**

**DESCRIPTION**

Instructs the transaction to ignore its outbound proxy when sending requests. In some cases you will want the transaction to ignore its outbound proxy event if the transaction 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 RvSipTransactionIgnoreOutboundProxy(
   IN RvSipTranscHandle hTransc,
   IN RvBool bIgnoreOutboundProxy);
```

**PARAMETERS**

- **hTransc**
  The transaction handle.

- **bIgnoreOutboundProxy**
  RV_TRUE if you wish the transaction to ignore its configured outbound proxy. RV_FALSE otherwise.

**RETURN VALUES**

Returns RvStatus.
RvSipTransactionSendToFirstRoute()

DESCRIPTION
Sets the loose route parameter for sending a request. If the transaction sends the message to a loose route proxy (the next hop proxy), the bSendToFirstRoute parameter will set to RV_TRUE. If the transaction sends the message to a strict route proxy (the next hop proxy), the bSendToFirstRoute parameter will set to RV_FALSE.

SYNTAX
RvStatus RvSipTransactionSetLooseRoute(
    IN RvSipTranscHandle hTransaction,
    IN RvBool bSendToFirstRouteHeader);

PARAMETERS

hTransaction
The transaction in which to set the value for bSendToLooseRoute.

bSendToFirstRouteHeader
The bSendToFirstRoute value to set in the transaction.

RETURN VALUES
Returns RvStatus.
RvSipTransactionDetachOwner()

DESCRIPTION
Detaches a transaction from the transaction owner. The owner will no longer be informed of transaction events and the transaction will use its default behavior. This function will cause termination of the transaction if the transaction is in one of the following states:

- RVSIP_TRANSC_STATE_SERVER_GEN_REQUEST_RCVD
- RVSIP_TRANSC_STATE_SERVER_INVITE_REQUEST_RCVD
- RVSIP_TRANSC_STATE_SERVER_CANCEL_REQUEST_RCVD
- RVSIP_TRANSC_STATE_MSG_SEND_FAILURE

SYNTAX
RvStatus RvSipTransactionDetachOwner(
    IN RvSipTranscHandle hTransaction);

PARAMETERS
hTransaction
The transaction from which to detach the owner.

RETURN VALUES
Returns RvStatus.
The Transaction Get and Set functions are:

- `RvSipTransactionGetNewMsgElementHandle()`
- `RvSipTransactionSetCallId()`
- `RvSipTransactionGetCallId()`
- `RvSipTransactionSetFromHeader()`
- `RvSipTransactionGetFromHeader()`
- `RvSipTransactionSetToHeader()`
- `RvSipTransactionGetToHeader()`
- `RvSipTransactionSetCSeqStep()`
- `RvSipTransactionGetCSeqStep()`
- `RvSipTransactionGetMethodStr()`
- `RvSipTransactionGetResponseCode()`
- `RvSipTransactionGetCurrentState()`
- `RvSipTransactionSetKeyFromMsg()`
- `RvSipTransactionGetCancelTransc()`
- `RvSipTransactionGetCancelledTransc()`
- `RvSipTransactionSetViaBranch()`
- `RvSipTransactionGetRequestUri()`
- `RvSipTransactionSetOwner()`
- `RvSipTransactionGetOwner()`
- `RvSipTransactionGet100RelStatus()`
- `RvSipTransactionGetReceivedMsg()`
- `RvSipTransactionIsUAC()`
- `RvSipTransactionGetAppHandle()`
- `RvSipTransactionSetAppHandle()`
- `RvSipTransactionGetOutboundMsg()`
- `RvSipTransactionSetPersistency()`
- `RvSipTransactionGetPersistency()`
- `RvSipTransactionSetConnection()`
- `RvSipTransactionGetConnection()`
- `RvSipTranscGetStackInstance()`
- `RvSipTransactionGetTransmitter()`
- `RvSipTransactionSetIsProxyFlag()`
Transaction Get and Set Functions

- RvSipTransactionSetTimers()
- RvSipTransactionSetRejectStatusCodeOnCreation()
- RvSipTransactionSetForceOutboundAddrFlag()
- RvSipTransactionGetReceivedFromAddress()
RvSipTransactionGetNewMsgElementHandle()

DESCRIPTION
Allocates a new message element on the transaction page, and returns the new element handle. The application may use this function to allocate a message header or a message address. It should then fill the element information, and set it back to the transaction using the relevant Set function. The function supports the following elements:
- To, From—you should set these headers back with RvSipTransactionSetToHeader() or RvSipTransactionSetFromHeader().
- Address of any type—you should supply the address to RvSipTransactionRequest().

SYNTAX
RvStatus RvSipTransactionGetNewMsgElementHandle(
    IN RvSipTranscHandle hTransc,
    IN RvSipHeaderType eHeaderType,
    IN RvSipAddressType eAddrType,
    OUT void* *phMsgElement);

PARAMETERS
hTransc
The handle to the transaction.

eHeaderType
The type of header to allocate. RVSIP_HEADERTYPE_UNDEFINED should be supplied when allocating an address.

eAddrType
The type of the address to allocate. RVSIP_ADDRTYPE_UNDEFINED should be supplied when allocating a header.

phMsgElement
The handle to the newly created header or address.
Transaction Get and Set Functions
RvSipTransactionGetNewMsgElementHandle()

RETURN VALUES
Returns RvStatus.
RvSipTransactionSetCallId()

DESCRIPTION
Sets the Call-ID of the transaction.

SYNTAX
RvStatus RvSipTransactionSetCallId(
    IN RvSipTranscHandle hTransc,
    IN RvChar *strCallId);

PARAMETERS

hTransc
The transaction handle.

strCallId
The NULL terminating string with the new Call-ID.

RETURN VALUES
Returns RvStatus.
**RvSipTransactionGetCallId()**

**DESCRIPTION**
Copies the transaction Call-ID into a given buffer. If the buffer allocated by the application is insufficient, an RV_ERROR_INSUFFICIENT_BUFFER status is returned and the `actualSize` parameter will be set to the actual size of the Call-ID string in the transaction.

**SYNTAX**

```
RvStatus RvSipTransactionGetCallId(
    IN  RvSipTranscHandle   hTransc,
    IN  RvInt32              bufSize,
    OUT RvChar               strCallId,
    OUT RvInt32              *actualSize);
```

**PARAMETERS**

- **hTransc**
The transaction handle.

- **bufSize**
The size of the application buffer for the Call-ID.

- **strCallId**
An application allocated buffer.

- **actualSize**
The actual size of the Call-ID.

**RETURN VALUES**
Returns `RvStatus`. 
RvSipTransactionSetFromHeader()

**DESCRIPTION**
Sets the From header of the *transaction*.

**SYNTAX**
```
RvStatus RvSipTransactionSetFromHeader(
    IN RvSipTranscHandle hTransc,
    IN RvSipPartyHeaderHandle hFrom);
```

**PARAMETERS**
- **hTransc**
The *transaction* handle.
- **hFrom**
The handle to the From header to set in the *transaction*.

**RETURN VALUES**
Returns RvStatus.
RvSipTransactionGetFromHeader()

DESCRIPTION
Returns the From header associated with the transaction. Attempting to alter the From header after the transaction left the IDLE state might cause unexpected results.

SYNTAX
RvStatus RvSipTransactionGetFromHeader(
    IN  RvSipTranscHandle hTransc,
    OUT RvSipPartyHeaderHandle *phFrom);

PARAMETERS

hTransc
The handle to the transaction.

phFrom
The pointer to the transaction From header handle.

RETURN VALUES
Returns RvStatus.
RvSipTransactionSetToHeader()

**DESCRIPTION**

Sets the To header of the *transaction*.

**SYNTAX**

```c
RvStatus RvSipTransactionSetToHeader(
    IN RvSipTranscHandle hTransc,
    IN RvSipPartyHeaderHandle hTo);
```

**PARAMETERS**

- **hTransc**
  
  The *transaction* handle.

- **hTo**
  
  The handle to the To header to set in the *transaction*.

**RETURN VALUES**

Returns *RvStatus*. 
RvSipTransactionGetToHeader()

DESCRIPTION

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

SYNTAX

RvStatus RvSipTransactionGetToHeader(
    IN   RvSipTranscHandle       hTransc,
    OUT  RvSipPartyHeaderHandle *phTo);

PARAMETERS

hTransc
The handle to the transaction.

phTo
A pointer to the transaction to the header handle.

RETURN VALUES

Returns RvStatus.
RvSipTransactionSetCSeqStep()

DESCRIPTION
Sets the CSeq-Step number of the transaction.

SYNTAX
RvStatus RvSipTransactionSetCSeqStep(
    IN RvSipTranscHandle hTransc,
    IN RvInt32 CSeqStep);

PARAMETERS
hTransc
The transaction handle.

CSeqStep
The CSeq step to be set in the transaction.

RETURN VALUES
Returns RvStatus.

REMARKS
If you do not set a CSeq step in the transaction, the Stack will generate a random CSeq.
**RvSipTransactionGetCSeqStep()**

**DESCRIPTION**

Gets the CSeq-step number of the transaction.

**SYNTAX**

```c
RvStatus RvSipTransactionGetCSeqStep(
    IN  RvSipTranscHandle hTransc,
    OUT RvInt32    *pCSeqStep);
```

**PARAMETERS**

- **hTransc**
  
The transaction from which to get CSeq-step value.

- **pCSeqStep**
  
The CSeq-step number of the transaction.

**RETURN VALUES**

Returns RvStatus.
**RvSipTransactionGetMethodStr()**

**DESCRIPTION**

Returns the *transaction* method as a string.

**SYNTAX**

```c
RvStatus RvSipTransactionGetMethodStr(
    IN RvSipTranscHandle hTransc,
    IN  RvUint32     bufferSize,
    OUT RvChar       *strMethodBuffer);
```

**PARAMETERS**

- **hTransc**
  
  The *transaction* from which to get the method.

- **bufferSize**
  
  The size of the buffer.

- **strMethodBuffer**
  
  The buffer to be filled with the method.

**RETURN VALUES**

Returns *RvStatus*. 
RvSipTransactionGetResponseCode()

DESCRIPTION

Gets the response code of the transaction or 0 if there is no relevant response code (if a response has not yet been sent or received).

SYNTAX

RvStatus RvSipTransactionGetResponseCode(
   IN RvSipTranscHandle hTransc,
   OUT RvInt16 *pResponseCode);

PARAMETERS

hTransc

The transaction from which to get response code.

pResponseCode

The transaction response code.

RETURN VALUES

Returns RvStatus.
RvSipTransactionGetCurrentState()

DESCRIPTION

Gets the current state of the transaction.

SYNTAX

RvStatus RvSipTransactionGetCurrentState(
    IN RvSipTranscHandle hTransc,
    OUT RvSipTransactionState *peState);

PARAMETERS

hTransc

The transaction handle.

peState

The current state of the transaction.

RETURN VALUES

Returns RvStatus.
RvSipTransactionSetKeyFromMsg()

DESCRIPTION
Initializes the transaction key from a given message. The following fields are set in the transaction:
- To and From (including tags)
- Call-ID
- CSeq step

SYNTAX
RvStatus RvSipTransactionSetKeyFromMsg(
    IN RvSipTranscHandle hTransc,
    IN RvSipMsgHandle hMsg);

PARAMETERS
hTransc
The transaction handle.

hmsg
The handle to the message from which the transaction key is set.

RETURN VALUES
Returns RvStatus.

REMARKS
- Before calling this function you should call RvSipTranscMgrCreateTransaction to create a new transaction.
- This function is normally used by proxy implementations. Before a proxy server can proxy a request, it should:
  1. Create a new client transaction with RvSipTranscMgrCreateTransaction().
  2. Get the received message from the server transaction using the RvSipTransactionGetReceivedMsg() function.
3. Set the *transaction* key using the 
RvSipTransactionSetKeyFromMsg() function.
The proxy can then use the initialized client *transaction* to 
proxy the request.
**RvSipTransactionGetCancelTransc()**

**DESCRIPTION**

Gets the handle to the *transaction* that cancels the *transaction* given as a parameter. When calling the **RvSipTransactionCancel()** function, a new CANCEL *transaction* is created. The **RvSipTransactionGetCancelTransc()** function returns the handle to this *transaction*. If the *transaction* given as a parameter was never cancelled, NULL is returned.

**SYNTAX**

```c
RvStatus RvSipTransactionGetCancelTransc(
    IN RvSipTranscHandle hTransc,
    OUT RvSipTranscHandle *phCancelTransc);
```

**PARAMETERS**

- **hTransc**
  
The *transaction* handle.

- **phCancelTransc**
  
The cancel *transaction* handle.

**RETURN VALUES**

Returns **RvStatus**.
RvSipTransactionGetCancelledTransc()

DESCRIPTION

Gets the handle to the transaction that a cancel transaction cancelled. You can call this function when CANCEL is received to get the handle to the cancelled transaction. If there is not cancelled transaction, NULL is returned.

SYNTAX

RvStatus RvSipTransactionGetCancelledTransc(
    IN RvSipTranscHandle hTransc,
    OUT RvSipTranscHandle *phCancelledTransc);

PARAMETERS

hTransc

The cancel transaction handle.

phCancelledTransc

The transaction cancelled by the cancel transaction.

RETURN VALUES

Returns RvStatus.
**RvSipTransactionSetViaBranch()**

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

**SYNTAX**
```c
RvStatus RvSipTransactionSetViaBranch(
    IN RvSipTranscHandle hTransc,
    IN RvChar *strBranch);
```

**PARAMETERS**
- **hTransc**
  The transaction handle.
- **strBranch**
  The Via branch as a string.

**RETURN VALUES**
Returns RvStatus.
RvSipTransactionGetRequestUri()

**DESCRIPTION**
Returns the Request URI that the *transaction* is using.

**SYNTAX**
```
RvStatus RvSipTransactionGetRequestUri(
    IN RvSipTranscHandle hTransc,
    OUT RvSipAddressHandle *hReqUri);
```

**PARAMETERS**
- **hTransc**
  The *transaction* handle.
- **hReqUri**
  The Request URI.

**RETURN VALUES**
Returns RvStatus.
RvSipTransactionSetOwner()

DESCRIPTION
Sets the owner of a given transaction. You can call this function for application transactions only.

SYNTAX
RvStatus RvSipTransactionSetOwner(
    IN RvSipTranscHandle hTransc,
    IN RvSipTranscOwnerHandle hOwnerHandle)

PERMEATES
hTransc
The transaction handle.

hOwnerHandle
The new transaction owner handle.

RETURN VALUES
Returns RvStatus.
RvSipTransactionGetOwner()

DESCRIPTION
Returns the handle to the owner of a given transaction.

SYNTAX
RvStatus RvSipTransactionGetOwner(
    IN RvSipTranscHandle hTransc,
    OUT RvSipTranscOwnerHandle *phOwnerHandle);

PARAMETERS
hTransc
The transaction handle.

phOwnerHandle
The transaction owner handle.

RETURN VALUES
Returns RvStatus.
RvSipTransactionGet100RelStatus()

**DESCRIPTION**

Returns the 100rel option tag status of a received INVITE request. A *transaction* can get a Request message with a Supported=100rel or Require=10rel headers. These headers indicate whether a reliable provisional response should be sent.

**SYNTAX**

```c
RvStatus RvSipTransactionGet100RelStatus(
    IN RvSipTranscHandle hTransc,
    OUT RvSipTransaction100RelStatus *relStatus);
```

**PARAMETERS**

- **hTransc**
  
  The *transaction* handle.

- **relStatus**
  
  The reliable status received in the INVITE request.

**RETURN VALUES**

Returns RvStatus.

**REMARKS**

This function can be used only for server INVITE transactions. For any other transaction, RVSIP_TRANSC_100_REL_UNDEFINED is returned.
RvSipTransactionGetReceivedMsg()

DESCRIPTION

Gets the message that was received by the transaction. You can call this function from the state changed call back function when the new state indicates that a message was received. If there is no valid received message, NULL is returned.

SYNTAX

RvStatus RvSipTransactionGetReceivedMsg(
    IN RvSipTranscHandle hTransc,
    OUT RvSipMsgHandle *phMsg);

PARAMETERS

hTransc

The transaction handle.

phMsg

A pointer to the received message.

RETURN VALUES

Returns RvStatus.
RvSipTransactionIsUAC()

DESCRIPTION
Returns RV_TRUE if the transaction is a client transaction or RV_FALSE if the transaction is a server transaction.

SYNTAX
RvStatus RvSipTransactionIsUAC(
    IN  RvSipTranscHandle hTransc,
    OUT RvBool   *bIsUAC);

PARAMETERS

hTransc
The transaction handle.

bIsUAC
The type of the transaction (RV_TRUE if the transaction is a client transaction, and RV_FALSE if the transaction is a server transaction).

RETURN VALUES
Returns RvStatus.
**RvSipTransactionGetAppHandle()**

**DESCRIPTION**

Returns the application handle to the given *transaction*.

**SYNTAX**

```c
RvStatus RvSipTransactionGetAppHandle(
    IN RvSipTranscHandle hTransc,
    OUT RvSipAppTranscHandle *phAppHandle)
```

**PARAMETERS**

- **hTransc**
  
  The *transaction* handle.

- **phAppHandle**
  
  The application handle.

**RETURN VALUES**

Returns *RvStatus*. 
Transaction Get and Set Functions
RvSipTransactionSetAppHandle()

**RvSipTransactionSetAppHandle()**

**DESCRIPTION**
Sets the application handle to the given *transaction*.

**SYNTAX**
```
RvStatus RvSipTransactionSetAppHandle(
    IN RvSipTranscHandle hTransc,
    IN RvSipAppTranscHandle hAppHandle)
```

**PARAMETERS**
- **hTransc**
  The *transaction* handle.
- **hAppHandle**
  The application handle.

**RETURN VALUES**
Returns *RvStatus*. 
RvSipTransactionGetOutboundMsg()

DESCRIPTION

Gets the message that is going to be sent by the transaction. You can call this function before calling an API function that causes a message to be sent, such as RvSipTransactionRequest(), RvSipTransactionRespond() and RvSipTransactionAck(), and in one of the following states:

- RVSIP_TRANSC_STATE_IDLE
- RVSIP_TRANSC_STATE_SERVER_INVITE_REQUEST/rcvd
- RVSIP_TRANSC_STATE_CLIENT_INVITE_FINAL_RESPONSE_RCVD
- RVSIP_TRANSC_STATE_SERVER_INVITE_REL_PROXY_RESPONSE_SENT
- RVSIP_TRANSC_STATE_SERVER_GEN_REQUEST_RCVD
- RVSIP_TRANSC_STATE_SERVER_CANCEL_REQUEST_RCVD

This function returns an empty message object. You can fill the message object with headers and a body and then call the API function. The transaction will add more headers to the same message object and the message will be sent.

Note  You must not use this function to add headers that are part of the transaction key, such as To, From, Call-ID and Via. Doing so will cause unexpected behavior of the transaction.

Note  If your application is multithreaded, you must lock your application object before calling the RvSipTransactionGetOutboundMsg() function and unlock it only after calling the API function that sends the message.

SYNTAX

RvStatus RvSipTransactionGetOutboundMsg(
    IN  RvSipTranscHandle hTransc,
    OUT RvSipMsgHandle  *phMsg)
Transaction Get and Set Functions
RvSipTransactionGetOutboundMsg()

PARAMETERS

hTransc
The transaction handle.

phMsg
A pointer to the message.

RETURN VALUES
Returns RvStatus.
RvSipTransactionSetPersistency()

DESCRIPTION
Changes the transaction persistency definition at runtime. This function receives a Boolean value that indicates whether or not the application wishes this transaction to be persistent. A persistent transaction will try to locate a suitable connection in the connection hash before opening a new connection. A connection that was constructed by a persistent transaction will automatically be inserted into the connection hash.

SYNTAX
RvStatus RvSipTransactionSetPersistency(
    IN RvSipTranscHandle hTransc,
    IN RvBool bIsPersistent);

PARAMETERS
hTransc
The transaction handle.

bIsPersistent
Determines the transaction persistency definition. RV_TRUE indicates that the transaction is persistent. Otherwise, RV_FALSE.

RETURN VALUES
Returns RvStatus.
RvSipTransactionGetPersistency()

DESCRIPTION
Returns the transaction persistency definition.

SYNTAX
RvStatus RvSipTransactionGetPersistency(
    IN RvSipTranscHandle hTransc,
    OUT RvBool *pbIsPersistent);

PARAMETERS
hTransc
The transaction handle.

pbIsPersistent
The transaction persistency definition. RV_TRUE indicates that the transaction is persistent. Otherwise, RV_FALSE.

RETURN VALUES
Returns RvStatus.
Transaction Get and Set Functions

RvSipTransactionSetConnection()

DESCRIPTION

Sets a connection to be used by the transaction. The supplied connection will be used only if it fits the local and remote addresses of the transaction. Otherwise, the transaction will either locate a suitable connection in the connection hash or create a new connection.

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

SYNTAX

RvStatus RvSipTransactionSetConnection(
    IN RvSipTranscHandle hTransaction,
    IN RvSipTransportConnectionHandle hConn);

PARAMETERS

hTransaction
The handle to the transaction.

hConn
The handle to the connection.

RETURN VALUES

Returns RvStatus.
RvSipTransactionGetConnection()

DESCRIPTION

Returns the connection that the transaction is currently using.

Note Only persistent transactions keep the currently-used connection. NULL will be returned if the transaction is not persistent.

SYNTAX

RvStatus RvSipTransactionGetConnection(
    IN RvSipTranscHandle hTransc,
    OUT RvSipTransportConnectionHandle *phConn);

PARAMETERS

hTransc
The handle to the transaction.

phConn
The handle to the currently used connection.

RETURN VALUES

Returns RvStatus.
**RvSipTranscGetStackInstance()**

**DESCRIPTION**

Returns the handle to the Stack instance to which this *transaction* belongs.

**SYNTAX**

```c
RvStatus RvSipTranscGetStackInstance(  
    IN RvSipTranscHandle hTransc,  
    OUT void* *phStackInstance)
```

**PARAMETERS**

- **hTransc**

  The handle to the *transaction*.

- **phStackInstance**

  A valid pointer which will be updated with a handle to the Stack instance.

**RETURN VALUES**

Returns *RvStatus*. 
**RvSipTransactionGetTransmitter()**

**DESCRIPTION**

Returns the transmitter that the transaction is using. The application can use the transmitter to hold and resume message sending. After address resolution is completed, the transaction calls the RvSipTransactionFinalDestResolvedEv() callback. At this stage, the application can get the transmitter from the transaction and call the RvSipTransmitterHoldSending() function. As a result, the message will not be sent to the remote party. The application can then manipulate the DNS list and the destination address. It is the responsibility of the application to call the RvSipTransmitterResumeSending() function so that the message will be sent to the updated destination.

**SYNTAX**

```c
RvStatus RvSipTransactionGetTransmitter(
    IN RvSipTranscHandle hTransc,
    OUT RvSipTransmitterHandle *phTrx)
```

**PARAMETERS**

- **hTransc**
  
  The transaction handle.

- **phTrx**
  
  The transmitter handle.

**RETURN VALUES**

Returns RvStatus.
RvSipTransactionSetIsProxyFlag()

DESCRIPTION
Sets the “isProxy” flag of the transaction. This flag defines whether or not the transaction should behave as a proxy transaction. The “isProxy” flag is always set according to the SIP Stack configuration when the transaction is created. If the SIP Stack is configured as a proxy, the “isProxy” flag will be set to RV_TRUE automatically.

A proxy application should call this function if it wishes to initiate a transaction by itself. For server transactions, you must not call this function from the RvSipTransactionCreateEv() callback. (You can use the RvSipTransactionStateChangedEv() callback).

SYNTAX
RvStatus RvSipTransactionSetIsProxyFlag(
    IN RvSipTranscHandle hTransc,
    IN RvBool bIsProxy);

PARAMETERS
hTransc
The transaction handle.

bIsProxy
The value to set in the “isProxy” flag.

RETURN VALUES
Returns RvStatus.
Transaction Get and Set Functions
RvSipTransactionSetTimers()

**RvSipTransactionSetTimers()**

**DESCRIPTION**

Sets new timer values to the transaction. According to RFC 3261, the transaction has to set various timers during its life cycle and perform various 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 SIP Stack configuration and are defined upon initialization. The application can use this function to change the different timer values of the transaction. The RvSipTimers structure received by this function contains all the configurable timers of the transaction.

You can set values to the timers that you want to configure. You can set UNDEFINED to timers that you wish the SIP 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 SIP Stack 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 various transaction timers. However, the application can set a different number and change the retransmission count.

**SYNTAX**

```c
RvStatus RvSipTransactionSetTimers(
    IN RvSipTranscHandle hTransc,
    IN RvSipTimers *pTimers,
    IN RvInt32 sizeOfTimersStruct);
```

**PARAMETERS**

**hTransc**

The handle to the transaction.

**pTimers**

A pointer to the structure that contains all the timeout values.
sizeOfTimersStruct
The size of the RvSipTimers structure.

RETURN VALUES
Returns RvStatus.
Transaction Get and Set Functions
RvSipTransactionSetRejectStatusCodeOnCreation()

**RvSipTransactionSetRejectStatusCodeOnCreation()**

**DESCRIPTION**

This function can be used synchronously from the `RvSipTransactionCreateEv()` callback when the application decides not to handle the transaction. A transaction that is not handled by the application is handled automatically by the SIP Stack and the SIP Stack rejects the request that created the transaction with a default response code.

The application can change the default response code by calling the `RvSipTransactionSetRejectStatusCodeOnCreation()` function and supplying a new response code. Note that after the application decides not to handle the transaction, it will not get any further callbacks that relate to this transaction. The application will not get the `RvSipTransactionMsgToSendEv()` callback for the reject response message or the TERMINATED state for the transaction.

**Note** When this function is used to reject a request, the application cannot use the outbound message mechanism to add information to the outgoing response message. If you wish to change the response message, you must use the regular respond mechanism.

**SYNTAX**

```c
RvStatus RvSipTransactionSetRejectStatusCodeOnCreation(
    IN RvSipTranscHandle hTransc,
    IN RvUint16 rejectStatusCode)
```

**PARAMETERS**

**hTransc**

The handle to the transaction.

**rejectStatusCode**

The reject status code for rejecting the request that created this object. The value must be between 300 and 699.
Transaction Get and Set Functions
RvSipTransactionSetRejectStatusCodeOnCreation()

**RETURN VALUES**

Returns RvStatus.
RvSipTransactionSetForceOutboundAddrFlag()

DESCRIPTION
Sets the force outbound address flag. This flag forces the transaction to send request messages to the outbound address regardless of the message content.

SYNTAX
RvStatus RvSipTransactionSetForceOutboundAddrFlag(
    IN RvSipTranscHandle hTransc,
    IN RvBool bForceOutboundAddr);

PARAMETERS

hTransc
The transaction handle.

bForceOutboundAddr
The flag value to set.

RETURN VALUES
Returns RvStatus.
**RvSipTransactionGetReceivedFromAddress()**

**DESCRIPTION**

Gets the address from which the transaction received its last message.

**SYNTAX**

```plaintext
RvStatus RvSipTransactionGetReceivedFromAddress(
    IN RvSipTranscHandle hTransaction,
    OUT RvSipTransportAddr *pAddr,
    OUT RvSipTransportAddrOptions *pOptions)
```

**PARAMETERS**

- **hTransaction**
  The handle to the transaction.

- **pAddr**
  Basic details about the received From address.

- **pOption**
  Options about the address details. If NULL, only basic data will be retrieved.

**RETURN VALUES**

Returns RvStatus.
The Transaction Authentication Functions are:

- RvSipTransactionAuthBegin()
- RvSipTransactionAuthProceed()
- RvSipTransactionRespondUnauthenticated()
- RvSipTransactionRespondUnauthenticatedDigest()
RvSipTransactionAuthBegin()

DESCRIPTION
Starts the server authentication process.

SYNTAX
RvStatus RvSipTransactionAuthBegin(
    IN RvSipTranscHandle hTransc);

PARAMETERS
hTransc
The transaction handle.

RETURN VALUES
Returns RvStatus.
Transaction Authentication Functions
RvSipTransactionAuthProceed()

RvSipTransactionAuthProceed()

DESCRIPTION
Orders the SIP Stack to proceed with the authentication process.

After the application received the RvSipTransactionAuthCredentialsFoundEv() callback, it checks the Authorization header that the SIP Stack found, and when it knows how to handle this header, the application calls this function and instructs the SIP Stack how to continue with the authentication process.

This information is given to the SIP Stack in the action parameter. The action options are as follows:

- **RVSIP_TRANSC_AUTH_ACTION_USE_PASSWORD**
  Checks the given Authorization header with the given password.

- **RVSIP_TRANSC_AUTH_ACTION_SUCCESS**
  The user checked the Authorization header and it is correct. (This will cause AuthCompletedEv to be called, with Success status.)

- **RVSIP_TRANSC_AUTH_ACTION_FAILURE**
  The user wants to stop the loop that searches for Authorization headers. (This will cause AuthCompletedEv to be called, with Failure status.)

- **RVSIP_TRANSC_AUTH_ACTION_SKIP**
  Orders to skip the given header, and continue the authentication procedure with the next header, if it exists. (This will cause AuthCredentialFoundEv to be called, or AuthCompletedEv with Failure status, if there are no more Authorization headers.)

SYNTAX

```c
RvStatus RvSipTransactionAuthProceed(
    IN RvSipTranscHandle hTransc,
    IN RvSipTransactionAuthAction action,
    IN RvSipAuthorizationHeaderHandle hAuthorization,
    IN RvChar *password);
```
Transaction Authentication Functions
RvSipTransactionAuthProceed()

PARAMETERS

hTransc
The transaction handle.

action
Specifies which action to take to proceed with the authentication process.

hAuthorization
handle to the Authorization header for which the function will check authentication. (This is needed if the action is RVSIP_TRANSC_AUTH_ACTION_USE_PASSWORD, or else NULL.)

password
The password for the realm+userName in the header.
(This is needed if action is RVSIP_TRANSC_AUTH_ACTION_USE_PASSWORD, or else NULL.)

RETURN VALUES
Returns RvStatus.
RvSipTransactionRespondUnauthenticated()

DESCRIPTION
Sends 401/407 response and adds a copy of the given authentication header to the response message.

SYNTAX
RvStatus RvSipTransactionRespondUnauthenticated(
    IN RvSipTranscHandle hTransc,
    IN RvUint16 responseCode,
    IN RvChar *strReasonPhrase,
    IN RvSipHeaderType headerType,
    IN void* hHeader);

PARAMETERS
hTransc
The transaction handle.

responseCode
The response code—401 or 407.

headerType
The type of the given header

hHeader
A pointer to the header to be set in the message.

RETURN VALUES
Returns RvStatus.
RvSipTransactionRespondUnauthenticatedDigest()

DESCRIPTION
Sends 401/407 response. This function adds the authentication header to the response message, with the given parameters.

SYNTAX
RvStatus RvSipTransactionRespondUnauthenticatedDigest(
    IN RvSipTranscHandle hTransc,
    IN RvUint16 responseCode,
    IN RvChar *strReasonPhrase,
    IN RvChar *strRealm,
    IN RvChar *strDomain,
    IN RvChar *strNonce,
    IN RvChar *strOpaque,
    IN RvBool bStale,
    IN RvSipAuthAlgorithm eAlgorithm,
    IN RvChar *strAlgorithm,
    IN RVSipAuthQopOption eQop,
    IN RvChar *strQop);

PARAMETERS

hTransc
The transaction handle.

responseCode
The response code—401 for a user agent or 407 for proxies.

strReasonPhrase
The reason phrase for this response code.

strRealm
The realm is mandatory.
Transaction Authentication Functions

**strDomain**
An optional string that may be NULL.

**strNonce**
An optional string that may be NULL.

**strOpaque**
An optional string that may be NULL.

**bstale**
TRUE or FALSE.

**eAlgorithm**
The enumeration of the algorithm. If RVSIP_AUTH_ALGORITHM_OTHER, the algorithm value is taken from the next argument.

**strAlgorithm**
The string of the algorithm. This parameter will be set only if the eAlgorithm parameter is set to be RVSIP_AUTH_ALGORITHM_OTHER.

**eQop**
The enumeration of Quality-of-Protection (Qop). If RVSIP_AUTH_QOP_OTHER, the Qop value will be taken from the next argument.

**strQop**
The string of Quality-of-Protection (Qop). This parameter will be set only if the eQop parameter is set to be RVSIP_AUTH_QOP_OTHER.

**RETURN VALUES**
Returns RvStatus.
TRANSACTION DNS FUNCTIONS

The Transaction DNS functions are:

- RvSipTransactionDNSContinue()
- RvSipTransactionDNSGetList()
Transaction DNS Functions
RvSipTransactionDNSContinue()

RvSipTransactionDNSContinue()

DESCRIPTION

Creates new transaction and copies all parameters from the original transaction to the new transaction. When a transaction fails to send a request due to a timeout, network error, or 503 response, the transaction moves to the MSG_SEND_FAILURE state. In this state, the application can do one of the following:

- Clone the transaction and send the request to the next DNS-resolved IP with the cloned transaction.
- Give up on this request and terminate the transaction.

To retry sending the message, the application must first call the RvSipTransactionDNSContinue() function. Calling this function clones the failure transaction and updates the DNS list. (To re-send the request to the next IP address, use the RvSipTransactionRequest() function). The RvSipTransactionDNSContinue function also terminates the original transaction unless the transaction received a 503 response on INVITE. If the function fails, the original transaction will not be terminated.

SYNTAX

RvStatus RvSipTransactionDNSContinue(
    IN RvSipTranscHandle hOrigTransaction,
    IN RvSipTranscOwnerHandle hOwner,
    OUT RvSipTranscHandle *hNewTransaction)

PARAMETERS

hOrigTransaction

The source transaction handle.

owner

A pointer to the new transaction owner.

hNewTransaction

A pointer to the new transaction handler.
Transaction DNS Functions
RvSipTransactionDNSContinue()

**RETURN VALUES**

Returns `RvStatus`. 
RvSipTransactionDNSGetList()

DESCRIPTION
Retrieves the DNS list object from the transaction. You can use this function if you wish to view or change the DNS List.

SYNTAX
RvStatus RvSipTransactionDNSGetList(
    IN RvSipTranscHandle hTransaction,
    OUT RvSipTransportDNSListHandle *phDnsList)

PARAMETERS
hTransaction
The transaction handle.

phDnsList
The DNS list handle

RETURN VALUES
Returns RvStatus.
The Transaction Local and Remote Address functions are:

- `RvSipTransactionSetLocalAddress()`
- `RvSipTransactionGetLocalAddress()`
- `RvSipTransactionSetOutboundDetails()`
- `RvSipTransactionGetOutboundDetails()`
- `RvSipTransactionGetCurrentLocalAddress()`
- `RvSipTransactionGetCurrentDestAddress()`
### RvSipTransactionSetLocalAddress()

**DESCRIPTION**
Sets the local address from which the transaction will send outgoing requests. The SIP Stack 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 Stack 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 (transport type and address type). If several configured local addresses match the remote address characteristics, the first configured address is taken.

You can use RvSipTransactionSetLocalAddress() to force the transaction to choose a specific local address for a specific transport and address type. For example, you can force the transaction to use the second configured UDP/IPv4 local address. If the transaction 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 in the initialization of the SIP Stack. If you configured the SIP Stack to listen to a 0.0.0.0 local address, you must use the same notation here.

**SYNTAX**
```
RvStatus RvSipTransactionSetLocalAddress(
    IN RvSipTranscHandle hTransaction,
    IN RvSipTransport eTransportType,
    IN RvSipTransportAddressType eAddressType,
    IN RvChar *localAddress,
    IN RvUint16 localPort);
```

**PARAMETERS**

- **hTransaction**
  The handle to the transaction.
Transaction Local and Remote Address Functions
RvSipTransactionSetLocalAddress()

\textbf{eTransportType}

The transport type (UDP, TCP or TLS).

\textbf{eAddressType}

The address type (IP or IP6).

\textbf{localAddress}

The local address to be set to this transaction.

\textbf{localPort}

The local port to be set to this transaction.

\textbf{RETURN VALUES}

Returns \textit{RvStatus}.
RvSipTransactionGetLocalAddress()

DESCRIPTION

Gets the local address that the transaction will use to send outgoing requests to a destination that listens to a specific transport and address type. This is the address the user set using the RvSipTransactionSetLocalAddress() function. If no address was set, the function will return the default, first configured local address of the requested transport and address type.

SYNTAX

RvStatus RvSipTransactionGetLocalAddress(
    IN RvSipTranscHandle hTransc,
    IN RvSipTransport eTransportType,
    IN RvSipTransportAddressType eAddressType,
    OUT RvChar *localAddress,
    OUT RvUint16 *localPort);

PARAMETERS

hTransaction
The handle to the transaction.

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

eAddressType
The address type (IP or IP6).

localAddress
The local address this transaction is using.

localPort
The local port this transaction is using.
**Transaction Functions**

RvSipTransactionGetLocalAddress()

**RETURN VALUES**

Returns `RvStatus`. 
### RvSipTransactionSetOutboundDetails()

**DESCRIPTION**

Sets all outbound proxy details to the transaction. All details are supplied in the `RvSipTransportOutboundProxyCfg` structure that includes parameters such as the IP address or host name, transport and port 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 `RvSipTransactionIgnoreOutboundProxy()` function was called.

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

**SYNTAX**

```c
RvStatus RvSipTransactionSetOutboundDetails(
    IN RvSipTranscHandle hTransc,
    IN RvSipTransportOutboundProxyCfg *pOutboundCfg,
)
```

**PARAMETERS**

- **hTransc**
  The handle to the transaction.

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

- **sizeOfCfg**
  The size of the outbound proxy configuration structure.

**RETURN VALUES**

Returns `RvStatus`. 
RvSipTransactionGetOutboundDetails()

**DESCRIPTION**

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

**SYNTAX**

```c
RvStatus RvSipTransactionGetOutboundDetails(
    IN RvSipTranscHandle hTransc,
    IN RvInt32 sizeOfCfg,
    OUT RvSipTransportOutboundProxyCfg *pOutboundCfg)
```

**PARAMETERS**

- **hTransc**
  The handle to the *transaction*.

- **sizeOfCfg**
  The size of the configuration structure.

- **pOutboundCfg**
  A pointer to outbound proxy configuration structure.

**RETURN VALUES**

Returns `RvStatus`.
RvSipTransactionGetCurrentLocalAddress()

DESCRIPTION

Gets the local address that the transaction actively used for sending the outgoing requests. For server transactions, this function returns the local address that was used for the incoming request.

SYNTAX

RvStatus RvSipTransactionGetCurrentLocalAddress(
    IN RvSipTranscHandle hTransc,
    OUT RvSipTransport *eTransportType,
    OUT RvSipTransportAddressType *eAddressType,
    OUT RvChar *localAddress,
    OUT RvUint16 *localPort);

PARAMETERS

hTransc
The handle to the transaction.

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

eAddressType
The address type (IP4 or IP6).

localAddress
The local address this transaction is using.

localPort
The local port this transaction is using.

RETURN VALUES

Returns RvStatus.
RvSipTransactionGetCurrentDestAddress()

DESCRIPTION

Gets the destination address that the transaction will use to send the next outgoing message. This address is available only in the context of the RvSipTransactionFinalDestResolvedEv() callback.

SYNTAX

RvStatus RvSipTransactionGetCurrentDestAddress(
    IN   RvSipTranscHandle   hTransc,
    OUT  RvSipTransport     *peTransportType,
    OUT  RvSipTransportAddressType   *peAddressType,
    OUT  RvChar              *strDestAddress,
    OUT  RvUint16            *pDestPort);

PARAMETERS

hTransc
The handle to the transaction.

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

peAddressType
The address type (IP4 or IP6).

strDestAddress
The destination IP address (must be of RVSIP_TRANSPORT_LEN_STRING_IP size).

pDestPort
The destination port.

RETURN VALUES

Returns RvStatus.
Transaction Local and Remote Address Functions
RvSipTransactionGetCurrentDestAddress()
TRANSMITTER MODULE

The Transmitter API of the Transmitter Module enables you to create transmitter objects (transmitter) and use them to send SIP and non-SIP messages.

This part includes the following section:

- Transmitter Functions
Transmitter API functions of the SIP Stack enable you to create and manage transmitter objects (*transmitters*), and use them to send SIP messages. Each *transmitter* can be used for sending a single SIP message. The *transmitter* performs the address resolution according to the message object and other parameters, such as the *transmitter* outbound address. The *transmitter* can also be used for sending non-SIP messages. This section contains the *transmitter* functions found in the *RvSipTransmitter.h* header file.

The functions included in this section are:

- Transmitter Manager Functions
- Transmitter Control Functions
- Transmitter Get and Set Functions
The Transmitter Manager (TransmitterMgr) is used mainly for creating new transmitters. The Transmitter Manager API Functions are:

- RvSipTransmitterMgrCreateTransmitter()
- RvSipTransmitterMgrSetAppMgrHandle()
- RvSipTransmitterMgrGetAppMgrHandle()
- RvSipTransmitterMgrGetStackInstance()
- RvSipTransmitterMgrSetEvHandlers()
RvSipTransmitterMgrCreateTransmitter()

**DESCRIPTION**

Creates a new *transmitter* and exchanges handles with the application. The new *transmitter* assumes the IDLE state and can be used for sending only one message. The message can be a SIP request or response, or a non-SIP message. After creating the *transmitter*, you can set different parameters to the *transmitter* with the Transmitter Set functions. You can then use `RvSipTransmitterSendMessage()` to send SIP messages to the remote party. In this case, the remote address is calculated from the message object along with the *transmitter* parameters. You can also use `RvSipTransmitterSendBuffer()` to send non-SIP messages. In this case you also need to supply the destination address. When creating a *transmitter* you should supply the *transmitter* event handlers to be notified of *transmitter* events.

**SYNTAX**

```c
RvStatus RvSipTransmitterMgrCreateTransmitter(
    IN RvSipTransmitterMgrHandle hTrxMgr,
    IN RvSipAppTransmitterHandle hAppTrx,
    IN RvSipTransmitterEvHandlers* pEvHandlers,
    IN RvInt32 sizeofEvHandlers,
    OUT RvSipTransmitterHandle* phTrx);
```

**PARAMETERS**

- **hTrxMgr**
  The handle to the *TransmitterMgr*.

- **hAppTrx**
  The application handle to the newly created *transmitter*.

- **pEvHandlers**
  The event handler structure for this *transmitter*.

- **sizeofEvHandlers**
  The size of the event handler structure.
Transmitter Manager Functions
RvSipTransmitterMgrCreateTransmitter()

phTrx
The handle to the newly created transmitter.

RETURN VALUES
Returns RvStatus.
**RvSipTransmitterMgrSetAppMgrHandle()**

**DESCRIPTION**

The application can have its own *Transmitter Mgr* handle. You can use the `RvSipTransmitterMgrSetAppMgrHandle()` function to save this handle in the SIP Stack *Transmitter Mgr*. At any given time, you can get this handle by calling the `RvSipTransmitterMgrGetAppMgrHandle()` function.

**SYNTAX**

```c
RvStatus RvSipTransmitterMgrSetAppMgrHandle(
    IN RvSipTransmitterMgrHandle hTrxMgr,
    IN void* pAppTransmitterMgr);
```

**PARAMETERS**

- **hTrxMgr**
  The handle to the SIP Stack *Transmitter Mgr*.

- **pAppTransmitterMgr**
  The application *Transmitter Mgr* handle.

**RETURN VALUES**

Returns *RvStatus*. 
Transmitter Manager Functions
RvSipTransmitterMgrGetAppMgrHandle()

**RvSipTransmitterMgrGetAppMgrHandle()**

**DESCRIPTION**

Returns the handle to the application TransmitterMgr. You set this handle in the SIP Stack using the RvSipTransmitterMgrSetAppMgrHandle() function.

**SYNTAX**

```c
RvStatus RvSipTransmitterMgrGetAppMgrHandle(
    IN RvSipTransmitterMgrHandle hTrxMgr,
    OUT void** pAppTransmitterMgr);
```

**PARAMETERS**

- **hTrxMgr**
  
  The handle to the SIP Stack TransmitterMgr.

- **pAppTransmitterMgr**
  
  The application TransmitterMgr handle.

**RETURN VALUES**

Returns RvStatus.
RvSipTransmitterMgrGetStackInstance()

DESCRIPTION
Returns the handle to the SIP Stack instance to which this TransmitterMgr belongs. The SIP Stack handle is returned as a void pointer. To use the SIP Stack handle, you should cast it to the RvSipStackHandle type.

SYNTAX
RvStatus RvSipTransmitterMgrGetStackInstance(
    IN RvSipTransmitterMgrHandle hTrxMgr,
    OUT void** phStackInstance);

PARAMETERS
hTrxMgr
The handle to the SIP Stack TransmitterMgr.

phStackInstance
The SIP Stack handle as a void pointer.

RETURN VALUES
Returns RvStatus.
RvSipTransmitterMgrSetEvHandlers()

DESCRIPTION
Sets the application event handlers for the TransmitterMgr. The application has to supply a structure that contains function pointers to the TransmitterMgr events to which it wishes to listen.

SYNTAX
RvStatus RvSipTransmitterMgrSetEvHandlers(
    IN RvSipTransmitterMgrHandle hTrxMgr,
    IN RvSipTransmitterMgrEvHandlers *pEvHandlers,
    IN RvUint32 evHandlerStructSize);

PARAMETERS

hTrxMgr
A handle to the TransmitterMgr.

pEvHandlers
A pointer to the structure containing application event handler pointers.

evHandlerStructSize
The size of the pEvHandlers structure.

RETURN VALUES
Returns RvStatus.
A transmitter is used for sending a single SIP message (request or response), or any non-SIP message. Proxies usually use transmitters to send Out of Context messages such as ACK, or responses that do not match any transaction.

The Transmitter Control functions are:

- RvSipTransmitterSendMessage()
- RvSipTransmitterHoldSending()
- RvSipTransmitterResumeSending()
- RvSipTransmitterTerminate()
- RvSipTransmitterSendBuffer()
Transmitter Control Functions
RvSipTransmitterSendMessage()

RvSipTransmitterSendMessage()

**DESCRIPTION**
Sends a SIP message to the remote party. The application should supply the message object that it wishes the transmitter to send. To send the message, the transmitter has to resolve the destination address. The transmitter first moves to the RESOLVING_ADDR state and starts the address resolution process. The transmitter calculates the remote address of the message according to RFC 3261 and RFC 3263, and takes into account the existence of the outbound proxy, Route headers and loose routing rules. Once address resolution is completed, the transmitter moves to the FINAL_DEST_RESOLVED state. This state is the last chance for the application to modify the Via header. The transmitter will then move to the READY_FOR_SENDING state and will try to send the message. If the message is sent successfully, the transmitter moves to the MSG_SENT state. If the transmitter fails to send the message, it will move to the MSG_SEND_FAILURE state. The RvSipTransmitterSendMessage() function can be called in two states:
- The IDLE state for initial sending
- The MSG_SEND_FAILURE state for sending the message to the next address in the DNS list, in case the previous address failed.

**SYNTAX**
```
RvStatus RvSipTransmitterSendMessage(
    IN RvSipTransmitterHandle hTrx,
    IN RvSipMsgHandle hMsgToSend,
    IN RvBool bHandleTopVia);
```

**PARAMETERS**

- **hTrx**
  The transmitter handle.

- **hMsgToSend**
  The message to send.
bHandleTopVia

Indicates whether or not the \textit{transmitter} should add a top Via header to Request messages and remove the top Via from response messages.

\textbf{RETURN VALUES}

Returns RvStatus.

\textbf{REMARKS}

- The DNS procedure is a-synchronous.
- If you wish the \textit{transmitter} to fix the top Via header of the message object according to the remote party address and transport types, you should first call the \texttt{RvSipTransmitterSetFixViaFlag()} function. Otherwise the \textit{transmitter} will only fix the Via transport and add the rport parameter in case it was configured to the SIP Stack. The send-by parameter will remain untouched.
Transmitter Control Functions
RvSipTransmitterHoldSending()

RvSipTransmitterHoldSending()

DESCRIPTION
Holds all sending activity of the transmitter and moves the transmitter to the ON_HOLD state. After address resolution is completed and before the message is sent, the transmitter moves to the FINAL_DEST_RESOLVED state. In this state, the application can hold the message sending by calling RvSipTransmitterHoldSending(). If the application wishes, it can change the remote address at this point using the RvSipTransmitterSetDestAddress() function and manipulate the rest of the DNS list using the Transport Layer API. If you wish the transmitter to use the next element in the list, you can use the RvSipTransmitterSetDestAddress() function to reset the current destination address. The transmitter will then repeat the address resolution process before sending the message. It is the responsibility of the application to resume the sending of the message using RvSipTransmitterResumeSending().

SYNTAX
RvStatus RvSipTransmitterHoldSending(
    IN RvSipTransmitterHandle hTrx);

PARAMETERS
hTrx
The transmitter handle.

RETURN VALUES
Returns RvStatus.
RvSipTransmitterResumeSending()

DESCRIPTION

Resumes the sending activities of the transmitter. This function can be called only in the ON_HOLD state of the transmitter. When this function is called, the transmitter first checks that a destination address exists. If so, the transmitter moves to the READY_FOR_SENDING state and then sends the message to this address. If there is no destination address (the user resets the address by calling RvSipTransmitterSetDestAddress() with NULL values), the transmitter returns to the RESOLVING_ADDR state and to the address resolution process.

SYNTAX

RvStatus RvSipTransmitterResumeSending(
       IN RvSipTransmitterHandle   hTrx);

PARAMETERS

hTrx

The transmitter handle.

RETURN VALUES

Returns RvStatus.
Transmitter Control Functions

RvSipTransmitterTerminate()

---

**RvSipTransmitterTerminate()**

**DESCRIPTION**

Terminates a transmitter and free all transmitter allocated resources. The transmitter then assumes the TERMINATED state.

**SYNTAX**

```c
RvStatus RvSipTransmitterTerminate(
    IN RvSipTransmitterHandle hTrx);
```

**PARAMETERS**

- **hTrx**
  
  The transmitter handle.

**RETURN VALUES**

Returns **RvStatus**.
RvSipTransmitterSendBuffer()

DESCRIPTION

Sends a buffer to a specified destination. Using this function, the application can send non-SIP messages to a desirable destination. Before calling this function, the application has to set the destination address to the transmitter using the RvSipTransmitterSetDestAddress() function. After the message is sent successfully, the transmitter assumes the MSG_SENT state. If the transmitter fails to send the message, it will assume the MSG_SEND_FAILURE state. In both cases, it is the responsibility of the application to terminate the transmitter.

Note  If this function returns with an error, the transmitter will not move to the MSG_SEND_FAILURE state.

SYNTAX

RvStatus RvSipTransmitterSendBuffer(
    IN RvSipTransmitterHandle hTrx,
    IN RvChar* strBuff,
    IN RvInt32 buffSize);

PARAMETERS

hTrx
The transmitter handle.

strBuff
The buffer to send.

buffSize
The buffer size.

RETURN VALUES

Returns RvStatus.
Transmitter Get and Set Functions

The Transmitter Get and Set functions are:

- `RvSipTransmitterSetLocalAddress()`
- `RvSipTransmitterGetLocalAddress()`
- `RvSipTransmitterSetOutboundDetails()`
- `RvSipTransmitterGetOutboundDetails()`
- `RvSipTransmitterSetPersistency()`
- `RvSipTransmitterGetPersistency()`
- `RvSipTransmitterSetConnection()`
- `RvSipTransmitterGetConnection()`
- `RvSipTransmitterSetFixViaFlag()`
- `RvSipTransmitterSetKeepMsgFlag()`
- `RvSipTransmitterSetAppHandle()`
- `RvSipTransmitterGetAppHandle()`
- `RvSipTransmitterGetCurrentLocalAddress()`
- `RvSipTransmitterGetDestAddress()`
- `RvSipTransmitterSetDestAddress()`
- `RvSipTransmitterSetIgnoreOutboundProxyFlag()`
- `RvSipTransmitterSetUseFirstRouteFlag()`
- `RvSipTransmitterSetViaBranch()`
- `RvSipTransmitterSetForceOutboundAddrFlag()`
Transmitter Functions

RvSipTransmitterSetLocalAddress()

DESCRIPTION

Sets the local address from which the transmitter will send outgoing messages. The SIP Stack 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 Stack sends an outgoing message, the local address (from where the message is sent) is chosen according to the characteristics of the remote address. Both the local and remote addresses must have the same characteristics (transport type and address type). If several configured local addresses match the remote address characteristics, the first configured address is taken. You can use RvSipTransmitterSetLocalAddress() to force the transmitter to choose a specific local address for a specific transport and address type. For example, you can force the transmitter to use the second configured UDP/IPv4 local address. If the transmitter sends a message 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 in the initialization of the SIP Stack. If you configured the SIP Stack to listen to a 0.0.0.0 local address, you must use the same notation here.

SYNTAX

RvStatus RvSipTransmitterSetLocalAddress(
    IN RvSipTransmitterHandle hTrx,
    IN RvSipTransport eTransportType,
    IN RvSipTransportAddressType eAddressType,
    IN RvChar* strLocalAddress,
    IN RvUint16 localPort);

PARAMETERS

hTrx

The handle to the transmitter.
Transmitter Get and Set Functions
RvSipTransmitterSetLocalAddress()

\textbf{eTransportType}
The transport type (UDP, TCP or TLS).

\textbf{eAddressType}
The address type (IPv4 or IPv6).

\textbf{strLocalAddress}
The local address to be set to this transmitter.

\textbf{localPort}
The local port to be set to this transmitter.

\textbf{RETURN VALUES}
Returns \texttt{RvStatus}.
RvSipTransmitterGetLocalAddress()

DESCRIPTION

Gets the local address which the transmitter will use to send outgoing messages to a destination that listens to a specific transport and address type. This is the address the user set using the RvSipTransmitterSetLocalAddress() function. If no address was set, the function will return the default, first configured local address of the requested transport and address type.

SYNTAX

RvStatus RvSipTransmitterGetLocalAddress(
    IN RvSipTransmitterHandle hTrx,
    IN RvSipTransport eTransportType,
    IN RvSipTransportAddressType eAddressType,
    OUT RvChar* strLocalAddress,
    OUT RvUint16* pLocalPort);

PARAMETERS

hTrx
The handle to the transmitter.

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

eAddressType
The address type (IPv4 or IPv6).

strLocalAddress
The local address this transmitter is using.

pLocalPort
The local port this transmitter is using.
Transmitter Get and Set Functions
RvSipTransmitterGetLocalAddress()

**RETURN VALUES**

Returns `RvStatus`. 
RvSipTransmitterSetOutboundDetails()

DESCRIPTION

Sets all outbound proxy details to the transmitter. All details are supplied in the RvSipTransportOutboundProxyCfg structure that includes parameters such as the IP address or host name, transport and port 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 RvSipTransmitterSetIgnoreOutboundProxyFlag() function was called.

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

SYNTAX

RvStatus RvSipTransmitterSetOutboundDetails(
    IN RvSipTransmitterHandle hTrx,
    IN RvSipTransportOutboundProxyCfg *pOutboundCfg,
    IN RvInt32 cfgStructSize);

PARAMETERS

hTrx
The handle to the transmitter.

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

sizeOfCfg
The size of the outbound proxy configuration structure.
Transmitter Get and Set Functions
RvSipTransmitterSetOutboundDetails()

**RETURN VALUES**

Returns *RvStatus.*
RvSipTransmitterGetOutboundDetails()

DESCRIPTION

Gets all the outbound proxy details that the transmitter uses. The details are placed in the RvSipTransportOutboundProxyCfg structure that includes parameters such as the IP address or host name, transport and port type.

If the outbound details were not set to the specific register-client but the outbound proxy was defined to the SIP Stack on initialization, the SIP Stack parameters will be returned. If the transmitter 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 RvSipTransmitterGetOutboundDetails(
    IN RvSipTransmitterHandle hTrx,
    IN RvInt32 cfgStructSize,
    OUT RvSipTransportOutboundProxyCfg *pOutboundCfg);

PARAMETERS

hTrx
The handle to the transmitter.

sizeOfCfg
The size of the configuration structure.

pOutboundCfg
A pointer to the outbound proxy configuration structure.

RETURN VALUES

Returns RvStatus.
Transmitter Get and Set Functions
RvSipTransmitterSetPersistency()

RvSipTransmitterSetPersistency()

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

SYNTAX
RvStatus RvSipTransmitterSetPersistency(
    IN RvSipTransmitterHandle hTrx,
    IN RvBool bIsPersistent);

PARAMETERS
hTrx
The transmitter handle.

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

RETURN VALUES
Returns RvStatus.
RvSipTransmitterGetPersistency()

**DESCRIPTION**

Returns the *transmitter* persistency definition.

**SYNTAX**

```c
RvStatus RvSipTransmitterGetPersistency(
    IN RvSipTransmitterHandle hTrx,
    OUT RvBool* pbIsPersistent);
```

**PARAMETERS**

- **hTrx**
  The *transmitter* handle.

- **pbIsPersistent**
  The *transmitter* persistency definition. RV_TRUE indicates that the *transmitter* is persistent. Otherwise, RV_FALSE.

**RETURN VALUES**

Returns RvStatus.
Transmitter Get and Set Functions
RvSipTransmitterSetConnection()

RvSipTransmitterSetConnection()

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

SYNTAX
RvStatus RvSipTransmitterSetConnection(
    IN RvSipTransmitterHandle hTrx,
    IN RvSipTransportConnectionHandle hConn);

PARAMETERS
hTrx
The handle to the transmitter.

hConn
The handle to the connection.

RETURN VALUES
Returns RvStatus.
**RvSipTransmitterGetConnection()**

**DESCRIPTION**

Returns the *connection* that is currently being used by the *transmitter*.

**SYNTAX**

```c
RvStatus RvSipTransmitterGetConnection(
    IN RvSipTransmitterHandle hTrx,
    OUT RvSipTransportConnectionHandle *phConn);
```

**PARAMETERS**

- **hTrx**
  The handle to the *transmitter*.

- **phConn**
  The handle to the currently used *connection*.

**RETURN VALUES**

Returns *RvStatus*. 
Transmitter Get and Set Functions
RvSipTransmitterSetFixViaFlag()

**RvSipTransmitterSetFixViaFlag()**

**DESCRIPTION**
Indicates that the transmitter should update the “sent-by” parameter of the top Via header before sending the message. The “sent-by” parameter should be updated according to the local address from which the request will be sent. This address is determined according to the remote address transport and address types. In many cases, the application does not know the remote party IP, transport, and address types in advance and therefore cannot know which local address will be used. In this case, the application might want the transmitter to update the top Via automatically and therefore should call the RvSipTransmitterSetFixViaFlag() function.

The default value of the `bFixVia` parameter is RV_FALSE and it should remain RV_FALSE if the application updates the top Via by itself.

**Note**  Regardless of the value of this parameter, the transmitter will update the transport and rport parameter of the top Via header.

**SYNTAX**

```c
RvStatus RvSipTransmitterSetFixViaFlag (  
    IN RvSipTransmitterHandle hTrx,  
    IN RvBool bFixVia);
```

**PARAMETERS**

- **hTrx**
The handle to the transmitter.

- **bFixVia**
RV_TRUE if the application wishes that the transmitter will update the “sent-by” parameter in the top Via header.

**RETURN VALUES**
Returns RvStatus.
RvSipTransmitterSetKeepMsgFlag()

DESCRIPTION
Indicates that the transmitter should not destruct the message immediately after encoding is completed. Before the transmitter sends a message, it first encodes the message object to a buffer and then sends the buffer to the remote party. After encoding is completed, the transmitter destructs the message object. In case of send failure, the transmitter moves to the MSG_SEND_FAILURE state. In this state the application can re-send the message object to the next address in the DNS list using RvSipTransmitterSendMessage(). The application can instruct the transmitter not to destruct the message after encoding by setting the KeepMsg flag to RV_TRUE and, in this case, it can supply NULL as a message handle to the RvSipTransmitterSendMessage() function. The message will be destructed only on termination of the transmitter.

SYNTAX
RvStatus RvSipTransmitterSetKeepMsgFlag (  
    IN RvSipTransmitterHandle hTrx,  
    IN RvBool bKeepMsg);  

PARAMETERS

hTrx
The handle to the transmitter.

bKeepMsg
RV_TRUE if the application wishes the message to be destructed on termination only.

RETURN VALUES
Returns RvStatus.
Transmitter Get and Set Functions
RvSipTransmitterSetAppHandle()

RvSipTransmitterSetAppHandle()

DESCRIPTION
Sets the transmitter application handle.

SYNTAX
RvStatus RvSipTransmitterSetAppHandle(
    IN RvSipTransmitterHandle hTrx,
    IN RvSipAppTransmitterHandle hAppTrx);

PARAMETERS
hTrx
The handle to the transmitter.

hAppTrx
A new application handle to the transmitter

RETURN VALUES
Returns RvStatus.
**RvSipTransmitterGetAppHandle()**

**DESCRIPTION**

Returns the application handle to this *transmitter*.

**SYNTAX**

```c
RvStatus RvSipTransmitterGetAppHandle ( 
    IN RvSipTransmitterHandle hTrx, 
    OUT RvSipAppTransmitterHandle *phAppTrx);
```

**PARAMETERS**

*hTrx*

The handle to the *transmitter*.

*phAppTrx*

The application handle to the *transmitter*.

**RETURN VALUES**

Returns *RvStatus*. 
RvSipTransmitterGetCurrentState()

DESCRIPTION
Gets the current state of the transmitter.

SYNTAX
RvStatus RvSipTransmitterGetCurrentState (  
    IN  RvSipTransmitterHandle hTrx,  
    OUT RvSipTransmitterState* peState);  

PARAMETERS
hTrx
The transmitter handle.

peState
The current state of the transmitter.

RETURN VALUES
Returns RvStatus.
RvSipTransmitterGetStackInstance()

DESCRIPTION
Returns the handle to the SIP Stack instance to which this transmitter belongs.

SYNTAX
RvStatus RvSipTransmitterGetStackInstance(
    IN RvSipTransmitterHandle hTrx,
    OUT void** phStackInstance);

PARAMETERS
hTrx
The handle to the transmitter.

phStackInstance
A void pointer which will be updated with the handle to the SIP Stack instance.

RETURN VALUES
Returns RvStatus.
Transmitter Get and Set Functions
RvSipTransmitterDNSGetList()

RvSipTransmitterDNSGetList()

DESCRIPTION
Retrieves the DNS list object from the transmitter. Each transmitter holds a DNS list object. The DNS list is updated in the address resolution process. The application can manipulate the list using the DNS List API functions.

SYNTAX
RvStatus RvSipTransmitterDNSGetList(
    IN RvSipTransmitterHandle hTrx,
    OUT RvSipTransportDNSListHandle* phDnsList);

PARAMETERS

hTrx
The transmitter handle.

phDnsList
The handle to the DNS list.

RETURN VALUES
Returns RvStatus.
**RvSipTransmitterGetCurrentLocalAddress()**

**DESCRIPTION**

Gets the local address the *transmitter* will use to send the message. This address is calculated according to the remote party transport and address types.

**SYNTAX**

```c
RvStatus RvSipTransmitterGetCurrentLocalAddress(
    IN RvSipTransmitterHandle hTrx,
    OUT RvSipTransport* eTransportType,
    OUT RvSipTransportAddressType* eAddressType,
    OUT RvChar* localAddress,
    OUT RvUint16* localPort);
```

**PARAMETERS**

- **hTrx**
  The handle to the *transmitter*.

- **eTransportType**
  The transport type (UDP, TCP or TLS).

- **eAddressType**
  The address type (IPv4 or IPv6).

- **localAddress**
  The local address this *transmitter* is using.

- **localPort**
  The local port this *transmitter* is using.

**RETURN VALUES**

Returns *RvStatus*. 
**RvSipTransmitterGetDestAddress()**

**DESCRIPTION**

Returns the destination address that the *transmitter* will use. When the *transmitter* assumes the FINAL_DEST_RESOLVED state, the application can get the destination address using the RvSipTransmitterGetDestAddress() function. To change this address, use the RvSipTransmitterSetDestAddress() function.

**SYNTAX**

```c
RvStatus RvSipTransmitterGetDestAddress(
    IN RvSipTransmitterHandle hTrx,
    IN RvInt32 addrStructSize,
    IN RvInt32 optionsStructSize,
    OUT RvSipTransportAddr* pDestAddr,
    OUT RvSipTransmitterExtOptions* pOptions);
```

**PARAMETERS**

- **hTrx**
  The *transmitter* handle.

- **addrStructSize**
  The size of the pDestAddr structure.

- **optionsStructSize**
  The size of the pOptions structure.

- **pDestAddr**
  The destination address that the *transmitter* will use.

- **pOptions**
  Advanced instructions for the message sending.
RETURN VALUES

Returns RvStatus.
**RvSipTransmitterSetDestAddress()**

**DESCRIPTION**
Sets the destination address that the transmitter will use. Use this function when you want the transmitter to use a specific address, regardless of the message content. If NULL is set as the destination address, the SIP Stack will continue the resolution process according to the DNS list.

**Note** Sending the transmitter for another “round” with the DNS can take time as DNS requests and responses are exchanged.

**SYNTAX**

```c
RvStatus RvSipTransmitterSetDestAddress(
    IN RvSipTransmitterHandle hTrx,
    IN RvSipTransportAddr* pDestAddr,
    IN RvInt32 addrStructSize,
    IN RvSipTransmitterExtOptions* pOptions,
    IN RvInt32 optionsStructSize);
```

**PARAMETERS**

- **hTrx**
  The transmitter handle.

- **addrStructSize**
  The size of the pDestAddr structure.

- **optionsStructSize**
  The size of the pOptions structure.

- **pDestAddr**
  The destination address that the transmitter will use. If set to NULL, the transmitter will continue the DNS procedure as if the destination IP was never retrieved.
Transmitter Get and Set Functions
RvSipTransmitterSetDestAddress()

pOptions
Advanced instructions for the message sending.

RETURN VALUES
Returns RvStatus.
**RvSipTransmitterSetIgnoreOutboundProxyFlag()**

**DESCRIPTION**

Instructs the *transmitter* to ignore its outbound proxy when sending requests. In some cases, the application will want the *transmitter* to ignore its outbound proxy event if it 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**

```c
RvStatus RvSipTransmitterSetIgnoreOutboundProxyFlag(
    IN RvSipTransmitterHandle hTrx,
    IN RvBool bIgnoreOutboundProxy);
```

**PARAMETERS**

- **hTrx**
  
The *transmitter* handle.

- **bIgnoreOutboundProxy**
  
  RV_TRUE if you wish the *transmitter* to ignored its configured outbound proxy. Otherwise, RV_FALSE.

**RETURN VALUES**

- Returns **RvStatus**.
RvSipTransmitterSetUseFirstRouteFlag()

Description
Indicates that a message should be sent according the first Route header and not according to the request URI. This will be the case when the next hop is a loose router.

Syntax
RvStatus RvSipTransmitterSetUseFirstRouteFlag(
    IN RvSipTransmitterHandle hTrx,
    IN RvBool bSendToFirstRoute);

Parameters
hTrx
The transmitter handle.

bSendToFirstRoute
RV_TRUE if the message should be sent to the first route. Otherwise, RV_FALSE.

Return Values
Returns RvStatus.
**RvSipTransmitterSetViaBranch()**

**DESCRIPTION**

Sets the branch parameter to the *transmitter*. The *transmitter* will add the branch to the top Via header of outgoing requests. If the message already has a branch, it will be replaced with the branch set by this function.

**SYNTAX**

```c
RvStatus RvSipTransmitterSetViaBranch(
    IN RvSipTransmitterHandle hTrx,
    IN RvChar* strViaBranch,
    IN RPOOL_Ptr* pRpoolViaBranch);
```

**PARAMETERS**

- **hTrx**
  
The *transmitter* handle.

- **strViaBranch**
  
The Via branch to add to the top Via header. This parameter is ignored for response messages.

- **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*. 
RvSipTransmitterSetForceOutboundAddrFlag()

DESCRIPTION
Sets the “force outbound addr” flag. This flag forces the transmitter to send request messages to the outbound address regardless of the message content.

SYNTAX
RvStatus RvSipTransmitterSetForceOutboundAddrFlag (  
    IN RvSipTransmitterHandle hTrx,  
    IN RvBool bForceOutboundAddr);  

PARAMETERS

hTrx
The handle to the transmitter.

bForceOutboundAddr
The flag value to set.

RETURN VALUES
Returns RvStatus.
Transmitter Get and Set Functions
RvSipTransmitterSetForceOutboundAddrFlag()
REGISTER-CLIENT MODULE

The Register-Client API enables you to register your User Agent to a registrar. The Register-Client API provides a set of callbacks that enable you to take part in the registration process.

This part includes the following section:

- Register-Client Functions
REGISTER-CLIENT FUNCTIONS

This section contains Register-Client API functions found in the
RvSipRegClient.h header file.

The functions included in this section are:

- Register-Client Manager Functions
- Register-Client Control Functions
- Register-Client Get And Set Functions
- Register-Client DNS Functions
- Register-Client Local and Remote Address Functions
- Register-Client Authentication Functions
Register-Client Manager Functions

The Register-client Manager (Register-clientMgr) manages all the register-client objects (register-clients). The Register-clientMgr is used to set the event handlers of the Register-Client module and to create new register-clients.

The Register-Client Manager API functions are:

- RvSipRegClientMgrSetEvHandlers()
- RvSipRegClientMgrSetCallId()
- RvSipRegClientMgrGetCallId()
- RvSipRegClientMgrCreateRegClient()
- RvSipRegClientMgrGetStackInstance()
**RvSipRegClientMgrSetEvHandlers()**

**DESCRIPTION**
Sets event handlers for all register-client events.

**SYNTAX**

```c
RvStatus RvSipRegClientMgrSetEvHandlers(
    IN RvSipRegClientMgrHandle hMgr,
    IN RvSipRegClientEvHandlers *pEvHandlers,
    IN RvInt32 structSize);
```

**PARAMETERS**

- **hMgr**
  The handle to the Register-clientMgr.

- **pEvHandlers**
  A pointer to the application event handler structure

- **structSize**
  The size of the event handler structure.

**RETURN VALUES**

Returns RvStatus.
**RvSipRegClientMgrSetCallId()**

**DESCRIPTION**

Sets the call-ID to the *Register-clientMgr*. The string is copied. The call-ID should be a NULL terminated string. The call-ID will be given to all *register-clients*. If you do not supply a call-ID, the *Register-clientMgr* will generate the call-ID for you.

**Note**  This function should be called only once after the initialization of the SIP Stack.

**SYNTAX**

```c
RvStatus RvSipRegClientMgrSetCallId(
    IN RvSipRegClientMgrHandle hMgr,
    IN RvChar *strCallId);
```

**PARAMETERS**

- **hMgr**
  The handle to the *Register-clientMgr*.

- **strCallId**
  The call-ID string.

**RETURN VALUES**

Returns *RvStatus*. 
RvSipRegClientMgrGetCallId()

DESCRIPTION

Gets the Call-ID of the Register-clientMgr. The Call-ID string is copied to strCallId. If the strCallId buffer is not adequate, the function returns RV_ERROR_INSUFFICIENT_BUFFER.

SYNTAX

RvStatus RvSipRegClientMgrGetCallId(
    IN RvSipRegClientMgrHandle hMgr,
    IN RvChar *strCallId,
    IN RvUint32 bufferSize);

PARAMETERS

hMgr
The handle to the Register-clientMgr.

strCallId
The application buffer. The Call-ID string is copied to this buffer.

bufferSize
The size of the strCallId buffer.

RETURN VALUES

Returns RvStatus.
Register-Client Manager Functions
RvSipRegClientMgrCreateRegClient()

RvSipRegClientMgrCreateRegClient()

**DESCRIPTION**
Creates a new register-client and replaces handles with the application. The new register-client assumes the IDLE state.

**To register with a Registrar**

1. Create a new Register-client object with this function.
2. Set, at least, the To and From headers and the Registrar address.
3. Call the RvSipRegClientRegister() function.

**SYNTAX**

```c
RvStatus RvSipRegClientMgrCreateRegClient(
    IN RvSipRegClientMgrHandle hRegClientMgr,
    IN RvSipAppRegClientHandle hAppRegClient,
    OUT RvSipRegClientHandle *phRegClient);
```

**PARAMETERS**

- **hRegClientMgr**
  The handle to the Register-clientMgr.

- **hAppRegClient**
  The application handle to the newly created register-client.

- **phRegClient**
  SIP Stack handle to the register-client.

**RETURN VALUES**

Returns RvStatus.
RvSipRegClientMgrGetStackInstance()

DESCRIPTION
Returns the handle to the SIP Stack instance to which this Register-clientMgr belongs.

SYNTAX
RvStatus RvSipRegClientMgrGetStackInstance(
    IN RvSipRegClientMgrHandle hRegClientMgr,
    OUT void* *phStackInstance)

PARAMETERS
hRegClientMgr
The handle to the Register-clientMgr.

phStackInstance
A valid pointer which will be updated with a handle to the SIP Stack instance.

RETURN VALUES
Returns RvStatus.
You can use the Register-Client API to request to register at a chosen registrar. The Register-Client functions are:

- RvSipRegClientRegister()
- RvSipRegClientTerminate()
- RvSipRegClientMake()
- RvSipRegClientDetachOwner()
- RvSipRegClientUseFirstRouteForRegisterRequest()
- RvSipRegClientDetachFromMgr()
- RvSipRegClientSetTranscTimers()
RvSipRegClientRegister()

DESCRIPTION
Sends a Register request to the registrar. The request URI To, From, Expires and Contact headers that were set to the register-client are inserted to the outgoing REGISTER Request message. The register-client then assumes the REGISTERING state and waits for a response from the server.

SYNTAX
RvStatus RvSipRegClientRegister(
    IN RvSipRegClientHandle hRegClient);

PARAMETERS

hRegClient
The register-client handle.

RETURN VALUES
Returns RvStatus.
RvSipRegClientTerminate()

DESCRIPTION
Terminates a register-client. This function destructs the register-client. You cannot reference the object after calling this function.

SYNTAX
RvStatus RvSipRegClientTerminate(
    IN RvSipRegClientHandle hRegClient);

PARAMETERS
hRegClient
The register-client handle.

RETURN VALUES
Returns RvStatus.
RvSipRegClientMake()

DESCRIPTION

After creating a register-client, you can use this function to set the To and From headers and the contact and Registrar addresses in the register-client, and send the REGISTER request. You can use this function if you have all the needed fields in textual format.

SYNTAX

RvStatus RvSipRegClientMake(
    IN RvSipRegClientHandle hRegClient,
    IN RvChar* strFrom,
    IN RvChar* strTo,
    IN RvChar* strRegistrar,
    IN RvChar* strContact);

PARAMETERS

hRegClient
The register-client handle.

strFrom
The initiator of the registration request. For example: “From: sip:user@home.com”.

strTo
The registering user. For example, “To: sip:bob@proxy.com”.

strRegistrar
The request URI of the registration request. This is the proxy address. For example: “sip:proxy.com”.

strContact
The location of the registering user. For example: “Contact: sip:bob@work.com”.
Register-Client Control Functions
RvSipRegClientMake()

**RETURN VALUES**
Returns *RvStatus.*
RvSipRegClientDetachOwner()

DESCRIPTION
Detaches the register-client owner. After calling this function, the application will stop receiving events for this register-client. This function can be called only when the object is in the TERMINATED state.

SYNTAX
RvStatus RvSipRegClientDetachOwner(
    IN RvSipRegClientHandle hRegClient)

PARAMETERS

hRegClient
The register-client handle.

RETURN VALUES
Returns RvStatus.
RvSipRegClientUseFirstRouteForRegisterRequest()

DESCRIPTION
An application may want to use a pre-loaded Route header when sending the register message. For this purpose, the application should add the Route headers to the outbound message, and call this function to notify the SIP Stack to send the request to the address of the first Route header in the outbound message.

SYNTAX
RvStatus RvSipRegClientUseFirstRouteForRegisterRequest(
    IN RvSipRegClientHandle hRegClient);

PARAMETERS
hRegClient
The handle to the reg-client object.

RETURN VALUES
Returns RvStatus.
RvSipRegClientDetachFromMgr()

DESCRIPTION

Detaches a register-client from the Register-clientMgr. By default, all the register-clients created in a single SIP Stack instance represent a single User Agent (UA) and therefore use the same Call-ID and an increased CSeq step.

The Register-clientMgr manages the Call-ID and CSeq step. When implementing a multi-lined UA, each register-client should have a different Call-ID (representing a different user) and should manage its own CSeq step counting. Calling RvSipRegClientDetachFromMgr() on a register-client will cause the register-client to generate its own Call-ID and manage its own CSeq step counter.

SYNTAX

RvStatus RvSipRegClientDetachFromMgr(
    IN RvSipRegClientHandle hRegClient);

PARAMETERS

hRegClient

The SIP Stack handle to the register-clients.

RETURN VALUES

Returns RvStatus.
RvSipRegClientSetTranscTimers()

DESCRIPTION
Sets timeout values for the transactions timers of the register-client. If some of the fields in pTimers are not set (UNDEFINED), this function will calculate the fields, or take the values from configuration.

SYNTAX
RvStatus RvSipRegClientSetTranscTimers(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipTimers *pTimers,
    IN RvInt32 sizeOfTimersStruct);

PARAMETERS
hRegClient
The handle to the register-client.

pTimers
A pointer to the structure containing all the timeout values.

sizeOfTimersStruct
The size of the RvSipTimers structure.

RETURN VALUES
Returns RvStatus.
The Get and Set functions are:

- RvSipRegClientSetFromHeader()
- RvSipRegClientGetFromHeader()
- RvSipRegClientSetToHeader()
- RvSipRegClientGetToHeader()
- RvSipRegClientSetContactHeader()
- RvSipRegClientGetFirstContactHeader()
- RvSipRegClientGetNextContactHeader()
- RvSipRegClientRemoveContactHeader()
- RvSipRegClientSetRegistrar()
- RvSipRegClientGetRegistrar()
- RvSipRegClientSetExpiresHeader()
- RvSipRegClientGetExpiresHeader()
- RvSipRegClientSetCallId()
- RvSipRegClientGetCallId()
- RvSipRegClientSetAppHandle()
- RvSipRegClientGetAppHandle()
- RvSipRegClientGetCurrentState()
- RvSipRegClientSetCSeqStep()
- RvSipRegClientGetCSeqStep()
- RvSipRegClientGetOutboundMsg()
- RvSipRegClientGetReceivedMsg()
- RvSipRegClientResetOutboundMsg()
- RvSipRegClientSetPersistency()
- RvSipRegClientGetPersistency()
- RvSipRegClientSetConnection()
- RvSipRegClientGetConnection()
- RvSipRegClientGetStackInstance()
- RvSipRegClientGetNewMsgElementHandle()
RvSipRegClientSetFromHeader()

DESCRIPTION
Sets the From header associated with the register-client.

SYNTAX
RvStatus RvSipRegClientSetFromHeader(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipPartyHeaderHandle hFrom);

PARAMETERS

hRegClient
The handle to the register-client.

hFrom
The handle to an application constructed and initialized From header.

RETURN VALUES
Returns RvStatus.
RvSipRegClientGetFromHeader()

DESCRIPTION
Returns the From header associated with the register-client. Attempting to alter the From address header after registration has been requested might cause unexpected results.

SYNTAX
RvStatus RvSipRegClientGetFromHeader(
    IN  RvSipRegClientHandle hRegClient,
    OUT RvSipPartyHeaderHandle *phFrom);

PARAMETERS
hRegClient
The handle to the register-client.

phFrom
A pointer to the register-client From header handle.

RETURN VALUES
Returns RvStatus.
Register-Client Get And Set Functions
RvSipRegClientSetToHeader()

RvSipRegClientSetToHeader()

DESCRIPTION
Sets the To header associated with the register-client.

SYNTAX
RvStatus RvSipRegClientSetToHeader(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipPartyHeaderHandle hTo);

PARAMETERS

hRegClient
The handle to the register-client.

hTo
The handle to an application constructed and initialized To header.

RETURN VALUES
Returns RvStatus.
RvSipRegClientGetToHeader()

DESCRIPTION
Returns the To header associated with the register-client. Attempting to alter the To header after Registration has been requested might cause unexpected results.

SYNTAX
RvStatus RvSipRegClientGetToHeader(
    IN  RvSipRegClientHandle hRegClient,
    OUT RvSipPartyHeaderHandle *phTo);

PARAMETERS
hRegClient
The handle to the register-client.

phTo
A pointer to the register-client To header handle.

RETURN VALUES
Returns RvStatus.
Register-Client Get And Set Functions
RvSipRegClientSetContactHeader()

**RvSipRegClientSetContactHeader()**

**DESCRIPTION**

Sets a contact header to the `register-client` Contact header list. Before calling `RvSipRegClientRegister()`, the application should use this function to supply all the Contact headers the application requires in order to register. These Contact headers are inserted into the Register Request message before being sent to the registrar.

**SYNTAX**

```c
RvStatus RvSipRegClientSetContactHeader(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipContactHeaderHandle hContactHeader);
```

**PARAMETERS**

- **hRegClient**
  The handle to the `register-client`.

- **hContactHeader**
  The handle to a Contact header to be set to the `register-client`.

**RETURN VALUES**

Returns `RvStatus`. 
RvSipRegClientGetFirstContactHeader()

**DESCRIPTION**

Gets the first Contact header of the Contact headers list.

**SYNTAX**

```c
RvSipContactHeaderHandle
*RvSipRegClientGetFirstContactHeader(
    IN RvSipRegClientHandle hRegClient);
```

**PARAMETERS**

- `hRegClient`
  The handle to the register-client.

**RETURN VALUES**

Returns a pointer to the first Contact header in the list. If the function returns NULL, there are no Contact headers in the list.
Register-Client Get And Set Functions  
RvSipRegClientGetNextContactHeader()  

---  

**RvSipRegClientGetNextContactHeader()**

**DESCRIPTION**

Gets a Contact header from the Contact headers list. The returned Contact header follows the Contact header indicated by phPrevContact in the list of Contact headers. For this purpose, the phPrevContact must be valid—a part of the list of contact headers.

**SYNTAX**

```c
RvSipContactHeaderHandle*
RvSipRegClientGetNextContactHeader(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipContactHeaderHandle *phPrevContact);
```

**PARAMETERS**

- **hRegClient**
  
  The handle to the register-client.

- **phPrevContact**
  
  A handle to the previous Contact header.

**RETURN VALUES**

Returns a pointer to the requested Contact header. If the function returns NULL, the requested Contact header does not exist.
RvSipRegClientRemoveContactHeader()

**DESCRIPTION**
Removes a Contact header from the Contact headers list.

**SYNTAX**

```c
RvStatus RvSipRegClientRemoveContactHeader(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipContactHeaderHandle hContact);
```

**PARAMETERS**

- **hRegClient**
The handle to the `register-client`.

- **hContact**
A handle to the Contact header to be removed.

**RETURN VALUES**
Returns `RvStatus`. 
**RvSipRegClientSetRegistrar()**

**DESCRIPTION**

Sets the SIP-URL of the registrar in the *register-client*. Before calling **RvSipRegClientRegister()**, your application should use this function to supply the SIP-URL of the registrar. The register request is sent to this SIP-URL.

You can change the registrar SIP-URL each time you call **RvSipRegClientRegister()**. You can change the registrar SIP-URL in the case of redirections and refreshes. The Registrar address must be a SIP-URL with no user name.

**SYNTAX**

```c
RvStatus RvSipRegClientSetRegistrar(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipAddressHandle hRequestUri);
```

**PARAMETERS**

- **hRegClient**
  The handle to the *register-client*.

- **hRequestUri**
  The handle to the registrar SIP-URL to be set to the *register-client*.

**RETURN VALUES**

Returns **RvStatus**.
RvSipRegClientGetRegistrar()

DESCRIPTION
Gets the SIP-URL of the registrar associated with the register-client.

SYNTAX
RvStatus RvSipRegClientGetRegistrar(
    IN RvSipRegClientHandle hRegClient,
    OUT RvSipAddressHandle *phRequestUri);

PARAMETERS
hRegClient
The handle to the register-client.

phRequestUri
The handle to the registrar SIP-URL.

RETURN VALUES
Returns RvStatus.
RvSipRegClientSetExpiresHeader()

DESCRIPTION
Sets an Expires header in the register-client. Before calling RvSipRegClientRegister(), the application can use this function to supply the required Expires header for use in the Register request. This Expires header is inserted into the Register Request message before being sent to the registrar.

SYNTAX
RvStatus RvSipRegClientSetExpiresHeader(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipExpiresHeaderHandle hExpiresHeader);

PARAMETERS
hRegClient
The handle to the register-client.

hExpiresHeader
The handle to an Expires header to be set to the register-client.

RETURN VALUES
Returns RvStatus.
RvSipRegClientGetExpiresHeader()

DESCRIPTION
Gets the Expires header associated with the register-client.

SYNTAX
RvStatus RvSipRegClientGetExpiresHeader(
    IN RvSipRegClientHandle hRegClient,
    OUT RvSipExpiresHeaderHandle *phExpiresHeader);

PARAMETERS

hRegClient
The handle to the register-client.

phExpiresHeader
The handle to the register-client Expires header.

RETURN VALUES
Returns RvStatus.
Register-Client Get And Set Functions
RvSipRegClientSetCallId()

RvSipRegClientSetCallId()

DESCRIPTION
Sets the register-client Call-ID. Calling this function is optional. If the Call-ID is not set, the register-client will take the Call-ID from the Register-clientMgr. A detached register-client will generate its own Call-ID. For more information on detached register-clients, see the RvSipRegClientDetachFromMgr() function.

SYNTAX
RvStatus RvSipRegClientSetCallId(
    IN RvSipRegClientHandle hRegClient,
    IN RvChar *strCallId);

PARAMETERS

hRegClient
The SIP Stack handle to the register-client.

strCallId
A NULL terminating string with the new Call-ID.

RETURN VALUES
Returns RvStatus.
RvSipRegClientGetCallId()

**DESCRIPTION**
Returns the Call-ID that the register-client uses. If the buffer that the application allocates is insufficient, an RV_ERROR_INSUFFICIENT_BUFFER status is returned and actualSize contains the size of the Call-ID string in the register-client.

**SYNTAX**
```c
RvStatus RvSipRegClientGetCallId (
    IN RvSipRegClientHandle hRegClient,
    IN RvInt32 bufSize,
    OUT RvChar *pstrCallId,
    OUT RvInt32 *actualSize);
```

**PARAMETERS**
- **hRegClient**
The SIP Stack handle to the register-client.
- **bufSize**
The size of the application buffer for the Call-ID.
- **strCallId**
An application-allocated buffer.
- **actualSize**
The actual size of the Call-ID.

**RETURN VALUES**
Returns RvStatus.
Register-Client Get And Set Functions
RvSipRegClientSetAppHandle()

RvSipRegClientSetAppHandle()

DESCRIPTION
Sets the register-client application handle. Usually the application replaces handles with the SIP Stack in the RvSipRegClientMgrCreateRegClient() function. The RvSipRegClientSetAppHandle() function is used if the application wishes to set a new application handle to the register-client.

SYNTAX
RvStatus RvSipRegClientSetAppHandle(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipAppRegClientHandle hAppRegClient);

PARAMETERS

hRegClient
The handle to the register-client.

hAppRegClient
A new application handle to set in the register-client.

RETURN VALUES
Returns RvStatus.
RvSipRegClientGetAppHandle()

DESCRIPTION

Gets the register-client application handle.

SYNTAX

RvStatus RvSipRegClientGetAppHandle(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipAppRegClientHandle *phAppRegClient);

PARAMETERS

hRegClient

The handle to the register-client.

hAppRegClient

The register-client application handle.

RETURN VALUES

Returns RvStatus.
Register-Client Get And Set Functions
RvSipRegClientGetCurrentState()

RvSipRegClientGetCurrentState()

DESCRIPTION
Returns the current state of the register-client.

SYNTAX
RvStatus RvSipRegClientGetCurrentState(
    IN   RvSipRegClientHandle   hRegClient,
    OUT  RvSipRegClientState   *peState);

PARAMETERS

hRegClient
The register-client handle.

peState
The register-client current state.

RETURN VALUES
Returns RvStatus.
RvSipRegClientSetCSeqStep()

**DESCRIPTION**
Sets the CSeq step associated with the *register-client*. The supplied CSeq must be larger than zero. If you do not set the CSeq step, the *register-client* will send the first Register request with CSeq=1 and will increase the CSeq for subsequent requests.

**SYNTAX**

```c
RvStatus RvSipRegClientSetCSeqStep(
    IN RvSipRegClientHandle hRegClient,
    IN RvUint32 cSeqStep);
```

**PARAMETERS**

- **hRegClient**
The *register-client* handle.
- **cSeqStep**
The CSeq step.

**RETURN VALUES**
Returns RvStatus.

**REMARKS**
- Most applications do not need to use this function. The *register-client* manages the CSeq step automatically.
- The CSeq that is supplied by using this function will be used only if the *register-client* detached from its Register-clientMgr.
**RvSipRegClientGetCSeqStep()**

**DESCRIPTION**

Returns the CSeq-Step associated with the register-client. The CSeq-Step is valid only after a registration request is successfully executed. Otherwise, the CSeq-Step is zero.

**SYNTAX**

```c
RvStatus RvSipRegClientGetCSeqStep(
    IN RvSipRegClientHandle hRegClient,
    OUT RvUint32 *pCSeqStep);
```

**PARAMETERS**

- **hRegClient**
  
The register-client handle.

- **pCSeqStep**
  
The register-client CSeq-Step.

**RETURN VALUES**

Returns RvStatus.
RvSipRegClientGetOutboundMsg()

DESCRIPTION
Gets the message that is going to be sent by the register-client. You can call this function before you call API functions such as RvSipRegClientRegister() and RvSipRegClientMake(). This function returns an empty message object. You can fill the message object with headers and a body.

Note You must not use this function to add headers that are part of the transaction key, such as To, From, Call-Id and Via. Doing so will cause unexpected behavior of the register-client.

Note If your application is multi-threaded, you must lock your application object before calling the SipRegClientGetOutboundMsg() function and unlock it only after calling the function that sends the message.

SYNTAX
RvStatus RvSipRegClientGetOutboundMsg(
   IN RvSipRegClientHandle hRegClient,
   OUT RvSipMsgHandle *phMsg)

PARAMETERS
hRegClient
The handle to the register-client.

phMsg
A pointer to the message.

RETURN VALUES
Returns RvStatus.
RvSipRegClientGetReceivedMsg()

DESCRIPTION

Gets the message that was received by the register-client. You can call this function from the RvSipRegClientStateChangedEv() callback function when the new state indicates that a message was received. If there is no valid received message, NULL is returned.

SYNTAX

RvStatus RvSipRegClientGetReceivedMsg(
    IN RvSipRegClientHandle hRegClient,
    OUT RvSipMsgHandle *phMsg)

PARAMETERS

hRegClient
The handle to the register-client.

phMsg
A pointer to the received message.

RETURN VALUES

Returns RvStatus.
RvSipRegClientResetOutboundMsg()

**DESCRIPTION**
Sets the outbound message of the register-client to NULL. If the register-client is about to send a message, it will create a new message to send.

**SYNTAX**

```c
RvStatus RvSipRegClientResetOutboundMsg(
    IN RvSipRegClientHandle hRegClient)
```

**PARAMETERS**

- **hRegClient**
  The handle to the register-client.

**RETURN VALUES**

Returns RvStatus.
Register-Client Get And Set Functions
RvSipRegClientSetPersistency()

RvSipRegClientSetPersistency()

DESCRIPTION
Changes the register-client persistency definition at runtime. This function receives a Boolean value that indicates whether or not the application wishes this register-client to be persistent. A persistent register-client will apply its persistency to the transactions it creates. A persistent transaction will try to locate a suitable connection in the connection hash before opening a new connection. A persistent register-client will also try to use the same connection for all outgoing requests by applying the last-used connection to the next-created transaction.

SYNTAX
RvStatus RvSipRegClientSetPersistency(
    IN RvSipRegClientHandle hRegClient,
    IN RvBool bIsPersistent);

PARAMETERS

hRegClient
The register-client handle.

bIsPersistent
Determines the register-client persistency definition. RV_TRUE indicates that the register-client is persistent. Otherwise, RV_FALSE.

RETURN VALUES
Returns RvStatus.
RvSipRegClientGetPersistency()

DESCRIPTION
Returns the register-client persistency definition.

SYNTAX
RvStatus RvSipRegClientGetPersistency(
    IN RvSipRegClientHandle hRegClient,
    OUT RvBool *pbIsPersistent);

PARAMETERS

hRegClient
The register-client handle.

pbIsPersistent
The register-client persistency definition. RV_TRUE indicates that the register-client is persistent. Otherwise, RV_FALSE.

RETURN VALUES
Returns RvStatus.
RvSipRegClientSetConnection()

DESCRIPTION

Sets a connection to be used by the register-client transactions. The register-client will hold this connection in its internal database. Whenever the register-client creates a new transaction, it supplies the transaction with the given connection. The transaction will use the connection only if it fits the local and remote addresses of the transaction. Otherwise, the transaction will either locate a suitable connection in the connection hash or create a new connection. The register-client will be informed that the supplied connection did not fit, and that a different connection was used and will update its database.

Note  You can set a connection only to a persistent register-client.

SYNTAX

RvStatus RvSipRegClientSetConnection(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipTransportConnectionHandle hConn);

PARAMETERS

hRegClient
The handle to the register-client.

hConn
The handle to the connection.

RETURN VALUES

Returns RvStatus.
RvSipRegClientGetConnection()

DESCRIPTION

Returns the connection that the register-client transactions are currently using.

Note  Only persistent register-clients keep the currently-used connection. NULL will be returned if the register-client is not persistent.

SYNTAX

RvStatus RvSipRegClientGetConnection(
    IN  RvSipRegClientHandle hRegClient,
    OUT RvSipTransportConnectionHandle *phConn);

DESCRIPTION

hRegClient
The handle to the register-client.

phConn
The handle to the currently used connection.

RETURN VALUES

Returns RvStatus.
RvSipRegClientGetStackInstance()

**DESCRIPTION**

Returns the handle to the SIP Stack instance to which this `register-client` belongs.

**SYNTAX**

```c
RvStatus RvSipRegClientGetStackInstance(
    IN RvSipRegClientHandle hRegClient,
    OUT void* *phStackInstance);
```

**PARAMETERS**

- **hRegClient**
  The handle to the `register-client`.

- **phStackInstance**
  A valid pointer that will be updated with a handle to the SIP Stack instance.

**RETURN VALUES**

Returns `RvStatus`.

Register-Client Get And Set Functions
RvSipRegClientGetNewMsgElementHandle()

RvSipRegClientGetNewMsgElementHandle()

DESCRIPTION
Allocates a new message element on the register-client page, and returns the new element handle. The application may use this function to allocate a message header or a message address. It should then fill the element information and set it back to the register-client using the relevant Set function. The function supports the following elements:

- Party—you should set these headers back with RvSipRegClientSetToHeader() or RvSipRegClientSetFromHeader().
- Contact—you should set this header back with RvSipRegClientSetContactHeader().
- Expires—you should set this header back with RvSipRegClientSetExpiresHeader().
- Authorization—you should set this header back with the header to RvSipRegClientSetInitialAuthorization() (for the IMS Add-on Module only).
- Address—you should supply the address to RvSipRegClientSetRegistrar().

Note You may use this function only in the register-client initialization stage, when the register-client is in the IDLE state. To change headers after the initialization stage, you must allocate the header on an application page, and then set it with the correct API function.

SYNTAX
RvStatus RvSipRegClientGetNewMsgElementHandle(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipHeaderType eHeaderType, 
    IN RvSipAddressType eAddrType, 
    OUT void* *phHeader);

Register-Client Functions 355
Register-Client Get And Set Functions

PARAMETERS

hRegClient
The handle to the register-client.

eHeaderType
The type of header to allocate. RVSIP_HEADERTYPE_UNDEFINED should be supplied when allocating an address.

eAddrType
The type of the address to allocate. RVSIP_ADDRTYPE_UNDEFINED should be supplied when allocating a header.

phHeader
The handle to the newly created header or address.

RETURN VALUES

Returns RvStatus.
The Register-Client DNS functions are:

- RvSipRegClientDNSGiveUp()
- RvSipRegClientDNSContinue()
- RvSipRegClientDNSReSendRequest()
- RvSipRegClientDNSGetList()
Register-Client DNS Functions

RvSipRegClientDNSGiveUp()

DESCRIPTION

Stops retrying to send a Register request after send failure. When a Register request fails due to a timeout, network error, or 503 response, the register-client moves to the MSG_SEND_FAILURE state. In this state, the application can do one of the following:

- Send the request to the next DNS-resolved IP.
- Give up on this request.

Calling RvSipRegClientDNSGiveUp() indicates that the application wishes to give up on this request. Retrying to send the request will stop and the register-client will change its state back to the previous state.

If this is the initial Register request of the register-client, calling RvSipRegClientDNSGiveUp() will terminate the register-client. If this is a refresh, the register-client will move back to the REGISTERED state.

SYNTAX

RvStatus RvSipRegClientDNSGiveUp(
    IN RvSipRegClientHandle hRegClient);

PARAMETERS

hRegClient

The handle to the register-client that sent the request.

RETURN VALUES

Returns RvStatus.
RvSipRegClientDNSContinue()

DESCRIPTION

Prepares the register-client to retry sending a request after the previous attempt failed. When a Register request fails due to a timeout, network error, or 503 response, the register-client moves to the MSG_SEND_FAILURE state. In this state, the application can do one of the following:

- Send the request to the next DNS-resolved IP.
- Give up on this request.

To retry sending the message, the application must first call the RvSipRegClientDNSContinue() function. Calling this function closes the failure transaction and updates the DNS list. (To re-send the request to the next IP address, use the RvSipRegClientDNSReSendRequest() function).

SYNTAX

RvStatus RvSipRegClientDNSContinue(
    IN RvSipRegClientHandle hRegClient);

PARAMETERS

hRegClient

The handle to the register-client that sent the request.

RETURN VALUES

Returns RvStatus.
### RvSipRegClientDNSReSendRequest()

#### DESCRIPTION
Re-sends a Register request after the previous attempt failed. When a Register request fails due to a timeout, network error, or 503 response, the register-client moves to the MSG_SEND_FAILURE state. In this state, the application can do one of the following:

- Send the request to the next DNS resolved IP.
- Give up on this request.

To re-send the Register request, first call RvSipRegClientDNSContinue(). You should then call RvSipRegClientDNSReSendRequest(). The request will automatically be sent to the next resolved IP address in the DNS list.

#### SYNTAX

```c
RvStatus RvSipRegClientDNSReSendRequest(
    IN RvSipRegClientHandle hRegClient);
```

#### PARAMETERS

- **hRegClient**
  The handle to the register-client that sent the request.

#### RETURN VALUES
Returns `RvStatus`. 
Register-Client DNS Functions
RvSipRegClientDNSGetList()

RvSipRegClientDNSGetList()

DESCRIPTION
Retrieves a DNS list object from the register-client current active transaction. When a Register request fails due to a timeout, network error, or 503 response, the register-client moves to the MSG_SEND_FAILURE state. In this state you can use RvSipRegClientDNSGetList() to get the DNS list if you wish to view or change it.

SYNTAX
RvStatus RvSipRegClientDNSGetList(
    IN  RvSipRegClientHandle hRegClient,
    OUT RvSipTransportDNSListHandle *phDnsList);

PARAMETERS

   hRegClient
   The handle to the register-client that sent the request.

   phDnsList
   The DNS list handle.

RETURN VALUES
Returns RvStatus.
Register-Client Local and Remote Address Functions

The Register-Client Local and Remote Address functions are:

- RvSipRegClientSetLocalAddress()
- RvSipRegClientGetLocalAddress()
- RvSipRegClientSetOutboundDetails()
- RvSipRegClientGetOutboundDetails()
Register-Client Local and Remote Address Functions

RvSipRegClientSetLocalAddress()

DESCRIPTION

Sets the local address from which the register-client will send outgoing requests. The SIP Stack 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/IPv6). When the SIP Stack 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 configured local addresses match the remote address characteristics, the first configured address is taken.

You can use RvSipRegClientSetLocalAddress() to force the register-client to choose a specific local address for a specific transport and address type. For example, you can force the register-client to use the second configured UDP/IPv4 local address. If the register-client 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 in the initialization of the SIP Stack. If you configured the SIP Stack to listen to a 0.0.0.0 local address, you must use the same notation here.

SYNTAX

RvStatus RvSipRegClientSetLocalAddress(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipTransport eTransportType,
    IN RvSipTransportAddressType eAddressType,
    IN RvChar *strLocalIPAddress,
    IN RvUint16 localPort);

PARAMETERS

hRegClient

The register-client handle.
Register-Client Local and Remote Address Functions
RvSipRegClientSetLocalAddress()

**eTransportType**
The transport type (UDP, TCP or TLS).

**eAddressType**
The address type (IPv4 or IPv6).

**strLocalIPAddress**
A string with the local address to be set to this register-client.

**localPort**
The local port to be set to this register-client. If you set the local port to 0, you will get a default value of 5060.

**RETURN VALUES**
Returns RvStatus.
RvSipRegClientGetLocalAddress()

DESCRIPTION

Gets the local address which the register-client will use to send outgoing requests to a destination with the supplied transport type and address type. This is the address the user set using the RvSipRegClientSetLocalAddress() function. If no address was set, the function will return the default, first configured local address of the requested transport and address type.

SYNTAX

RvStatus RvSipRegClientGetLocalAddress(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipTransport eTransportType,
    IN RvSipTransportAddressType eAddressType,
    OUT RvChar *strLocalIPAddress,
    OUT RvUint16 *pLocalPort);

PARAMETERS

hRegClient

The register-client handle.

eTransportType

The transport type (UDP, TCP or TLS).

eAddressType

The address type (IP4 or IP6).

strLocalIPAddress

The local address that the register-client will use for the supplied transport and address types. (The buffer must be longer than 48 bytes).

pLocalPort

The local port that the register-client will use.
Register-Client Local and Remote Address Functions
RvSipRegClientGetLocalAddress()

RETURN VALUES

Returns \texttt{RvStatus}. 
RvSipRegClientSetOutboundDetails()

DESCRIPTION
Sets all outbound proxy details to the register-client. All details are supplied in the RvSipTransportOutboundProxyCfg structure that includes parameters such as the IP address or host name, transport and port type. Requests sent by this object will use the outbound details specifications as a remote address.

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

SYNTAX
RvStatus RvSipRegClientSetOutboundDetails(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipTransportOutboundProxyCfg *pOutboundCfg,
    IN RvInt32 sizeOfCfg);

PARAMETERS

hRegClient
The handle to the register-client.

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

sizeOfCfg
The size of the outbound proxy configuration structure.

RETURN VALUES
Returns RvStatus.
RvSipRegClientGetOutboundDetails()

DESCRIPTION

Gets all outbound proxy details that the register-client uses. The details are placed in the RvSipTransportOutboundProxyCfg structure that includes parameters such as the IP address or host name, transport and port type. If the outbound details were not set to the specific register-client, but the outbound proxy was defined to the SIP Stack on initialization, the SIP Stack parameters will be returned. If the register-client 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 RvSipRegClientGetOutboundDetails(
   IN RvSipRegClientHandle hRegClient,
   IN RvInt32 sizeOfCfg,
   OUT RvSipTransportOutboundProxyCfg *pOutboundCfg);

PARAMETERS

hRegClient
The handle to the register-client.

sizeOfCfg
The size of the configuration structure.

pOutboundCfg
A pointer to the outbound proxy configuration structure.

RETURN VALUES

Returns RvStatus.
The Register-Client Authentication functions are:

- RvSipRegClientAuthenticate()
- RvSipRegClientGetCurrProcessedAuthObj()
- RvSipRegClientAuthObjGet()
- RvSipRegClientAuthObjRemove()
Register-Client Authentication Functions
RvSipRegClientAuthenticate()

**RvSipRegClientAuthenticate()**

**DESCRIPTION**
Re-sends the Register request with authentication information. This function should be called in the AUTHENTICATED state after receiving a 401 or 407 response from the server. The Authorization header is added to the outgoing request. The register-client then assumes the REGISTERING state and waits for a response from the server.

**SYNTAX**

```c
RvStatus RvSipRegClientAuthenticate(
    IN RvSipRegClientHandle hRegClient);
```

**PARAMETERS**

- **hRegClient**
The register-client handle.

**RETURN VALUES**

Returns RvStatus.
RvSipRegClientGetCurrProcessedAuthObj()

**DESCRIPTION**

Retrieves the *authentication-object* that is currently being processed by the *authenticator*. (for application usage in the RvSipAuthenticatorGetSharedSecretEv() callback).

**SYNTAX**

RvStatus RvSipRegClientGetCurrProcessedAuthObj (  
    IN RvSipRegClientHandle hRegClient,  
    OUT RvSipAuthObjHandle* phAuthObj);

**PARAMETERS**

**hRegClient**

The handle to the register-client.

**phAuthObj**

The *authentication-object* handle.

**RETURN VALUES**

Returns RvStatus.
Register-Client Authentication Functions
RvSipRegClientAuthObjGet()

RvSipRegClientAuthObjGet()

DESCRIPTION
Retrieves authentication-object from the list in the register-client. You may get the first, last or next object. If you get the next object, you must supply the current object in the relative parameter.

SYNTAX
RvStatus RvSipRegClientAuthObjGet(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipListLocation eLocation,
    IN RvSipAuthObjHandle hRelativeAuthObj,
    OUT RvSipAuthObjHandle* phAuthObj);

PARAMETERS
hRegClient
The handle to the register-client.

eLocation
The location in the list (first, next or last).

hRelativeAuthObj
The relative object in the list (relevant for the next location).

phAuthObj
The authentication-object handle.

RETURN VALUES
Returns RvStatus.
RvSipRegClientAuthObjRemove()

DESCRIPTION
Removes an authentication-object from the list in the register-client.

SYNTAX
RvStatus RvSipRegClientAuthObjRemove(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipAuthObjHandle hAuthObj);

PARAMETERS
hRegClient
The handle to the register-client.

hAuthObj
The authentication-object handle.

RETURN VALUES
Returns RvStatus.
Register-Client Authentication Functions
RvSipRegClientAuthObjRemove()
AUTHENTICATOR MODULE

The Authenticator API enables you to authenticate outgoing requests, and challenge and verify the credentials of incoming requests.

This part includes the following section:

- Authenticator Functions
This section contains Authenticator API functions found in the RvSipAuthenticator.h header file.
The functions included in this section are:

- Authenticator Control Functions
- Authenticator Get and Set Functions
Authenticator Control Functions

The Authenticator Control API functions included in this section are:

- RvSipAuthenticatorCredentialsSupported()
- RvSipAuthenticatorVerifyCredentials()
- RvSipAuthenticatorVerifyCredentialsExt()
- RvSipAuthenticatorAddProxyAuthorizationToMsg()
- RvSipAuthenticatorPrepareAuthorizationHeader()
- RvSipAuthenticatorPrepareAuthenticationInfoHeader()
**RvSipAuthenticatorCredentialsSupported()**

**DESCRIPTION**

This function is for server-side authentication. It checks parameters in the Authorization header, such as the Quality-of-Protection (Qop), algorithm and scheme, and determines if the SIP Stack can or cannot authenticate the header. If the SIP Stack cannot authenticate the header, the application must authenticate the Authorization header by itself.

**SYNTAX**

```c
RvStatus RvSipAuthenticatorServerValidityChecking(
    IN RvSipAuthenticatorHandle hAuth,
    IN RvSipAuthorizationHeaderHandle hAuthorization,
    OUT RvBool* bIsSupported);
```

**PARAMETERS**

- **hAuth**
  The handle to the authenticator.

- **hAuthorization**
  The handle to the Authentication header.

- **bIsSupported**
  RV_TRUE if supported.

**RETURN VALUES**

Returns RvStatus.
RvSipAuthenticatorVerifyCredentials()

DESCRIPTION

This function is for a server-side authentication. The application can verify the credentials received in an incoming Request message when using this function. The application supplies the password that belongs to username and realm found in the Authorization header, and wishes to know if the Authorization header is correct for this username. This function creates the digest string as specified in RFC 2617, and compares it to the digest string inside the given Authorization header. If it is equal, the header is correct.

SYNTAX

RvStatus RvSipAuthenticatorVerifyCredentials(
    IN RvSipAuthenticatorHandle hAuth,
    IN RvSipAuthorizationHeaderHandle hAuthorization,
    IN RvChar *password,
    IN RvChar *strMethod,
    OUT RvBool *isCorrect);

PARAMETERS

hAuth
The handle to the authenticator.

hAuthorization
The handle to the Authorization header.

password
The password of the user indicated in the Authorization header.

strMethod
The method type of the request.

isCorrect
RV_TRUE if correct, otherwise RV_FALSE.
Authenticator Control Functions

RvSipAuthenticatorVerifyCredentials()

**RETURN VALUES**

Returns *RvStatus*. 
RvSipAuthenticatorVerifyCredentialsExt()

DESCRIPTION

This function is for server-side authentication. By using this function, the application can verify the credentials received in an incoming request message. The application supplies the password that belong to the user name and realm found in the Authorization header, and wishes to know if the Authorization header is correct for this username.

This function creates the digest string as specified in RFC 2617, and compares it to the digest string inside the given Authorization header. If it is equal, the header is correct. In comparison to RvSipAuthenticatorVerifyCredentials(), this function supports “auth-int” Quality-of-Protection (Qop).

Syntax

RvStatus RvSipAuthenticatorVerifyCredentialsExt (
    IN RvSipAuthenticatorHandle hAuth,
    IN RvSipAuthenticationHeaderHandle hAuthorization,
    IN RvChar* password,
    IN RvChar* strMethod,
    IN void* hObject,
    IN void* peObjectType,
    IN RvSipMsgHandle hMsg,
    OUT RvBool* isCorrect);

PARAMETERS

hAuth

The handle to the authenticator.

hAuthenticationHeader

The handle to the Authorization header.

password

The password of the user in the hAuthorization header.
strMethod
The method type of the request.

hObject
The handle to the object to be authenticated.

peObjectType
The type of object to be authenticated.

hMsg
The message object to which $hAuthorization$ belongs.

isCorrect
RV_TRUE if correct, otherwise, RV_FALSE.

**RETURN VALUES**
Returns RvStatus.
Authenticator Control Functions
RvSipAuthenticatorAddProxyAuthorizationToMsg()

RvSipAuthenticatorAddProxyAuthorizationToMsg()

DESCRIPTION

Adds a Proxy-Authorization header to the supplied message object. You can use this function only if you set a Proxy-Authenticate header to the authenticator, using the RvSipAuthenticatorSetProxyAuthInfo() function. The authenticator uses the challenge found in the Proxy-Authenticate header to build the correct Proxy-Authorization header.

You should use this function if you want to add credentials to outgoing requests that were sent by stand-alone transactions. This function should be called from the RvSipTransactionMsgToSendEv() callback of the transaction. For other SIP Stack objects, the process of adding the Authorization header is automatic.

SYNTAX

RvStatus RvSipAuthenticatorAddProxyAuthorizationToMsg(
    IN RvSipAuthenticatorHandle hAuth,
    IN RvSipMsgHandle hMsg,
    IN RvInt32 nonceCount,
    IN void* hObject,
    IN RvSipCommonStackObjectType eObjectType);

PARAMETERS

hAuth

The handle to the authenticator.

hMsg

The handle to the message, to which the Proxy-Authorization header should be added.

nonceCount

The nonce-count value that should be used for the MD5 signature generation. If set to UNDEFINED, the internal nonce-count that is managed by the authenticator will be used. If no internal nonce-value was set (with the RvSipAuthenticatorSetNonceCount() function), 1 will be used.
Authenticator Control Functions
RvSipAuthenticatorAddProxyAuthorizationToMsg()

**hObject**

The handle to the object to which this message belongs. The handle will be supplied back to the application in some of the authenticator callback functions. You can supply a NULL value.

**eObjectType**

The type of object to which this message belongs. The type will be supplied back to the application in some of the authenticator callback functions. You can set this value to RVSIP_COMMON_STACK_OBJECT_TYPE_UNDEFINED.

**RETURN VALUES**

Returns RvStatus.
Authenticator Control Functions

RvSipAuthenticatorPrepareAuthorizationHeader()

DESCRIPTION

Prepares an Authorization/Proxy-Authorization header based on the WWW-
Authenticate/Authenticate header and other data, supplied as parameters.

SYNTAX

RvStatus RvSipAuthenticatorPrepareAuthorizationHeader(
    IN   RvSipAuthenticatorHandle    hAuth,
    IN   RvSipAuthenticationHeaderHandle
        hAuthenticationHeader,
    IN   RvChar*                    strMethod,
    IN   RvSipAddressHandle         hRequestUri,
    IN   RvInt32                    nonceCount,
    IN   void*                      hObject,
    IN   RvSipMsgHandle             hMsg,
    INOUT RvSipAuthorizationHeaderHandle
        hAuthorizationHeader);

PARAMETERS

hAuth

The handle to the authenticator.

hAuthenticationHeader

The handle to the WWW-Authenticate/Authenticate header.

strMethod

The string representing the SIP method name. This parameter can be NULL. If
NULL, the hMsg parameter should be supplied. The method name will be
extracted from the hMsg message.

hRequestUri

The handle to the address object representing the address to which the message
with the prepared Authorization/Proxy-Authorization header will be sent.
**Authenticator Control Functions**
RvSipAuthenticatorPrepareAuthorizationHeader()

**nonceCount**
For information about the nonce count, see RFC 3261.
If the value of this parameter is undefined, the global nonce count used. The application can set the global nonce count by calling RvSipAuthenticatorSetNonceCount(). If the nonce count is undefined and the global nonce count is not set, an error will occur.

**hObject**
The handle to the application object that will be supplied to the application as a callback parameter during Authorization header preparing. Can be NULL.

**hMsg**
The handle to the message that will contain the prepared Authorization/Proxy-Authorization header.

**hAuthorizationHeader**
The handle to the prepared header. If NULL, the new header will be constructed using the hMsg page.

**RETURN VALUES**
Returns RvStatus.
Authenticator Control Functions  
RvSipAuthenticatorPrepareAuthenticationInfoHeader()

**RvSipAuthenticatorPrepareAuthenticationInfoHeader()**

**DESCRIPTION**

Builds an Authentication-Info header by doing the following:

- Generates and sets the response-auth string which is similar to the “request-digest” in the Authorization header. (Note that the MD5 and shared-secret callback will be called during the response-auth generation).
- Sets the “message-qop”, “cnonce”, and “nonce-count” parameters if needed.
- Sets the given next-nonce parameter to the header.

This function receives an Authorization header (which was received in the request, and was successfully authorized), and uses its parameters to build the Authentication-info header. You can use this function to set only the next-nonce parameter, by supplying NULL in the Authorization header argument.

**SYNTAX**

```c
RvStatus RvSipAuthenticatorPrepareAuthenticationInfoHeader(
    IN RvSipAuthenticatorHandle hAuth,
    IN RvSipAuthorizationHeaderHandle hAuthorization,
    IN RvChar* nextNonce,
    IN RvSipCommonStackObjectType eObjType,
    IN void* hObject,
    IN RvSipMsgHandle hMsg,
    INOUT RvSipAuthenticationInfoHeaderHandle hAuthenticationInfo);
```

**PARAMETERS**

- **hAuth**
  
  The handle to the authenticator.

- **hAuthorization**
  
  The handle to the Authorization header.
nextNonce
The value for the next-nonce parameter. This value is optional.

eObjType
The type of the following hObject parameter.

hObject
The handle to the application object that will be supplied to the application as a callback parameter during preparation of the Authorization header. Can be NULL.

hMsg
The handle to the message that will contain the prepared Authentication-info header.

hAuthenticationInfo
The handle to the prepared header. Can be NULL. If NULL, the hMsg parameter should be provided and a new header will be constructed, while using the page of the message.

**RETURN VALUES**

Returns RvStatus.
The Authenticator Get and Set functions are:

- `RvSipAuthenticatorSetEvHandlers()
- `RvSipAuthenticatorSetAppHandle()
- `RvSipAuthenticatorGetStackInstance()
- `RvSipAuthenticatorSetProxyAuthInfo()
- `RvSipAuthenticatorSetNonceCount()
- `RvSipAuthObjGetAuthenticationHeader()
- `RvSipAuthObjSetUserInfo()
- `RvSipAuthObjSetAppContext()
- `RvSipAuthObjGetAppContext()`
RvSipAuthenticatorSetEvHandlers()

**DESCRIPTION**
Sets event handlers for authentication events. For information about the Authenticator callback functions, see Authenticator API Type Definitions.

**SYNTAX**

```c
RvStatus RvSipAuthenticatorSetEvHandlers(
    IN RvSipAuthenticatorHandle hAuth,
    IN RvSipAuthenticatorEvHandlers *evHandlers,
    IN RvUint32 evHandlersSize);
```

**PARAMETERS**

- **hAuth**
  - The handle to the authenticator.

- **evHandlers**
  - A pointer to event handlers structure.

- **evHandlersSize**
  - The size of the event handlers structure.

**RETURN VALUES**

Returns RvStatus.
RvSipAuthenticatorSetAppHandle()

DESCRIPTION
Sets the application authenticator handle to the SIP Stack authenticator. This handle is returned in some of the authenticator callback functions.

SYNTAX

\begin{verbatim}
RvStatus RvSipAuthenticatorSetAppHandle(
    IN RvSipAuthenticatorHandle hAuth, 
    IN RvSipAppAuthenticatorHandle hAppAuth);
\end{verbatim}

PARAMETERS

\begin{itemize}
\item \textbf{hAuth}
  The handle to the authenticator.
\item \textbf{hAppAuth}
  The handle to the application authenticator.
\end{itemize}

RETURN VALUES

Returns \texttt{RvStatus}. 
RvSipAuthenticatorGetStackInstance()

DESCRIPTION
Returns the handle to the SIP Stack instance to which this authenticator belongs.

SYNTAX
RvStatus RvSipAuthenticatorGetStackInstance(
    IN RvSipAuthenticatorHandle hAuth,
    OUT void* *phStackInstance);

PARAMETERS

hAuth
The handle to the authenticator.

phStackInstance
A valid pointer that will be updated with a handle to the SIP Stack instance.

RETURN VALUES
Returns RvStatus.
Authenticator Get and Set Functions
RvSipAuthenticatorSetProxyAuthInfo()

RvSipAuthenticatorSetProxyAuthInfo()

DESCRIPTION
Sets a Proxy-Authenticate header to the authenticator. This header will be used to authenticate requests in advance. The authenticator will use this header to build a Proxy-Authorization header that will be placed automatically in all requests sent by call-legs, register-clients and subscriptions. An application should use this function if it knows the challenge requested by its proxy. By setting the Proxy-Authorization header in initial requests, the application may avoid the first message exchange of a request and its 407 response.

SYNTAX
RvStatus RvSipAuthenticatorSetProxyAuthInfo(
    IN RvSipAuthenticatorHandle hAuth,
    IN RvSipAuthenticationHeaderHandle hAuthHeader,
    IN RvChar *strUserName,
    IN RvChar *strPassword);

PARAMETERS

hAuth
The handle to the authenticator.

hAuthHeader
The handle to the authentication header.

strUserName
The user name string.

strPassword
The password string.

RETURN VALUES
Returns RvStatus.
**REMARKS**

- If a SIP Stack object received a 407 response despite the existence of the Proxy-Authorization header in the request, the object will use the new, received challenge to create correct credentials.
- This function can be used only for setting a Proxy-Authenticate header. You cannot set a WWW-Authenticate header.
Authenticator Get and Set Functions
RvSipAuthenticatorSetNonceCount()

RvSipAuthenticatorSetNonceCount()

**DESCRIPTION**
Sets the initial value of the nonce-count parameter that will be used when creating the Proxy-Authorization header which is placed in outgoing requests. This nonce count is used with the challenge supplied by calling the RvSipAuthenticatorSetProxyAuthInfo() function for authenticating messages in advance. The nonce-count value is incremented by the authenticator after each header calculation, according to RFC 2617.

**SYNTAX**
RvStatus RvSipAuthenticatorSetNonceCount(
  IN RvSipAuthenticatorHandle hAuth,
  IN RvInt32 nonceCount);

**PARAMETERS**

_**hAuth**_
The handle to the authenticator.

_**nonceCount**_
The nonce count value to use.

**RETURN VALUES**
Returns RvStatus.

**REMARKS**
- The range of legal values for the nonce-count is [0,MAX_INT32].
- The value of the nonce count is reset each time the application calls the RvSipAuthenticatorSetProxyAuthInfo() function with a new nonce field.
**Authenticator Get and Set Functions**

**RvSipAuthObjGetAuthenticationHeader()**

**DESCRIPTION**

Returns the Authentication header kept in an authentication-object.

**SYNTAX**

```c
RvStatus RvSipAuthObjGetAuthenticationHeader(
    IN RvSipAuthObjHandle hAuthObj,
    OUT RvSipAuthenticationHeaderHandle* phHeader,
    OUT RvBool *pbIsValid);
```

**PARAMETERS**

- **hAuthObj**
  The handle to the authentication-object.

- **phHeader**
  The handle to the Authentication header in the authentication-object.

- **pbIsValid**
  Specifies whether or not the Authentication header is valid.

**RETURN VALUES**

Returns RvStatus.
Authenticator Get and Set Functions
RvSipAuthObjSetUserInfo()

**RvSipAuthObjSetUserInfo()**

**DESCRIPTION**
Sets the username and password in the *authentication-object*. An application can use this function to set the user information before sending the request. I

**SYNTAX**

```c
RvStatus RvSipAuthObjSetUserInfo(
    IN RvSipAuthObjHandle hAuthObj,
    IN RvChar* pstrUserName,
    IN RvChar* pstrUserPw);
```

**PARAMETERS**

- **hAuthObj**
  The handle to the *authentication-object*.

- **pstrUserName**
  A pointer to the user-name string. If NULL, will erase the existing parameter in the *authentication-object*.

- **pstrUserPw**
  A pointer to the password string. If NULL, will erase the existing parameter in the *authentication-object*.

**RETURN VALUES**

Returns *RvStatus*. 
RvSipAuthObjSetAppContext()

DESCRIPTION
Sets a pointer to the application context in the authentication-object.

SYNTAX
RvStatus RvSipAuthObjSetAppContext(
    IN RvSipAuthObjHandle hAuthObj,
    IN void* pContext);

PARAMETERS

hAuthObj
The handle to the authentication-object.

pContext
A pointer to the application context. If NULL, will erase the existing parameter in the authentication-object.

RETURN VALUES
Returns RvStatus.
RvSipAuthObjGetAppContext()

**DESCRIPTION**
Gets the pointer of the application context in the *authentication-object*. (The application set the context in RvSipAuthObjSetAppContext()).

**SYNTAX**

```c
RvStatus RvSipAuthObjGetAppContext(
    IN  RvSipAuthObjHandle    hAuthObj,
    OUT void*                *ppContext);
```

**PARAMETERS**

**hAuthObj**
The handle to the *authentication-object*.

**ppContext**
A pointer to the application context.

**RETURN VALUES**
Returns RvStatus.
The Event Notification API functions enable you to create subscriptions (subscriptions) and handle notifications related to the subscriptions. The Event Notification API provides a set of callbacks that enable you to take part in the event notification process.

This part includes the following section:

- Event Notification Functions
The Event Notification API functions of the SIP Stack enable you to create and manage subscription \( (\text{subscription}) \) and notification \( (\text{notification}) \) objects. The Event Notification functions can be used for implementing the event-notification feature. This section contains the Event Notification functions found in the \texttt{RvSipSubscription.h} header file.

The functions included in this section are:

- Subscription Manager API
- Subscription Control API
- Subscription Get and Set API
- Subscription Notification API
- Subscription Authentication API
- Subscription DNS API
- Subscription Refer API
- Subscription High Availability API
- Subscription Forking API
The Subscription Manager (SubscriptionMgr) is in charge of all the subscriptions. It is used to set the event handlers of the Subscription module and to create new subscriptions.

The Subscription Manager API functions are:

- `RvSipSubsMgrSetEvHandlers()`
- `RvSipSubsMgrCreateSubscription()`
- `RvSipSubsMgrCreateOutOfBandSubscription()`
- `RvSipSubsMgrGetStackInstance()`
- `RvSipSubsMgrGetAppMgrHandle()`
- `RvSipSubsMgrSetAppMgrHandle()`
RvSipSubsMgrSetEvHandlers()

**DESCRIPTION**
Sets event handlers for all subscription events.

**SYNTAX**

```
RvStatus RRvSipSubsMgrSetEvHandlers(
    IN RvSipSubsMgrHandle hMgr,
    IN RvSipSubsEvHandlers *pEvHandlers,
    IN RvInt32 structSize);
```

**PARAMETERS**

**hMgr**
The handle to the SubscriptionMgr.

**pEvHandlers**
A pointer to the structure containing application event handler pointers.

**structSize**
The size of the event handler structure.

**RETURN VALUES**
Returns RvStatus.
Subscription Manager API

RvSipSubsMgrCreateSubscription()

**DESCRIPTION**

Creates a new *subscription* and exchanges handles with the application. The new *subscription* assumes the IDLE state. You can supply a *call-leg* handle to create the *subscription* within an already established *call-leg*. (An established *call-leg* means that an INVITE request and 1XX or 2XX response were already sent/received, so the *call-leg* has a to-tag.) Otherwise, set NULL in the *call-leg* handle parameter. To establish a new active *subscription*:

1. Create a new *subscription* with this function.
2. Initialize the *subscription* parameters.
3. Call the RvSipSubsSubscribe() function.

**SYNTAX**

```c
RvStatus RvSipSubsMgrCreateSubscription(
    IN RvSipSubsMgrHandle hSubsMgr,
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppSubsHandle hAppSubs,
    OUT RvSipSubsHandle *phSubs);
```

**PARAMETERS**

- **hSubsMgr**
  The handle to the *SubscriptionMgr*.

- **hCallLeg**
  The handle to a *call-leg* that owns the *subscription*. If NULL, the *subscription* will create a new dialog, and will not be related to any *call-leg*.

- **hAppSubs**
  The application handle to the newly created *subscription*.

- **hSubs**
  The SIP Stack handle to the *subscription*. 
Subscription Manager API
RvSipSubsMgrCreateSubscription()

**Return Values**

Returns `RvStatus`. 
RvSipSubsMgrCreateOutOfBandSubscription()

DESCRIPTION

Creates a new out-of-band subscription and exchanges handles with the application. An out-of-band subscription is created with no SUBSCRIBE request. The new subscription assumes the ACTIVE state. You can supply a call-leg handle to create the subscription within an already established call-leg. (An established call-leg means that an INVITE request and 1XX or 2XX response were already sent/received, so the call-leg has a to-tag.) Otherwise, set NULL in the call-leg handle parameter.

To establish a new out-of-band subscription:

1. Create a new subscription with this function.
2. Initialize the subscription parameters.

Note If you do not supply a call-leg handle to this subscription in the initialization state, you must first set the dialog Call-ID parameter and then call the RvSipSubsInit() function.

SYNTAX

RvStatus RvSipSubsMgrCreateOutOfBandSubscription(
    IN RvSipSubsMgrHandle hSubsMgr,
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppSubsHandle hAppSubs,
    IN RvSipSubscriptionType eSubsType,
    OUT RvSipSubsHandle *hSubs);

PARAMETERS

hSubsMgr

The handle to the SubscriptionMgr.

hCallLeg

The handle to a call-leg that owns the subscription. If NULL, the subscription will create a new dialog and will not be related to any call-leg.
**Subscription Manager API**

RvSipSubsMgrCreateOutOfBandSubscription()

- **hAppSubs**
  The application handle to the newly created *subscription*.

- **eSubsType**
  The subscriber or notifier.

- **hSubs**
  The SIP Stack handle to the *subscription*. 
RvSipSubsMgrGetStackInstance()

DESCRIPTION
Returns the handle to the SIP Stack instance to which this SubscriptionMgr belongs.

SYNTAX
RvStatus RvSipSubsMgrGetStackInstance(
    IN RvSipSubsMgrHandle hSubsMgr,
    OUT void **phStackInstance);

PARAMETERS

hSubsMgr
The handle to the SubscriptionMgr.

phStackInstance
A valid pointer which will be updated with a handle to the SIP Stack instance.

RETURN VALUES
Returns RvStatus.
**RvSipSubsMgrGetAppMgrHandle()**

**DESCRIPTION**

Returns the handle to the SubscriptionMgr of the application. You set this handle in the SIP Stack using the **RvSipSubsMgrSetAppMgrHandle()** function.

**SYNTAX**

```c
RvStatus RvSipSubsMgrGetAppMgrHandle(
    IN RvSipSubsMgrHandle hSubsMgr,
    OUT void** pAppSubsMgr);
```

**PARAMETERS**

- **hSubsMgr**
  The handle to the SIP Stack SubscriptionMgr.

- **pAppSubsMgr**
  The SubscriptionMgr handle of the application.

**RETURN VALUES**

Returns **RvStatus**.
RvSipSubMgrSetAppMgrHandle()

DESCRIPTION
The application can have its own SubscriptionMgr handle. You can use this function to save this handle in the SIP Stack SubscriptionMgr.

SYNTAX
RvStatus RvSipSubMgrSetAppMgrHandle(
    IN RvSipSubMgrHandle hSubsMgr,
    IN void* pAppSubsMgr);

PARAMETERS
hSubsMgr
The handle to the Stack SubscriptionMgr.

pAppSubsMgr
The SubscriptionMgr handle of the application.
A subscription represents a SIP subscription as defined in draft sip-event-05. A subscription is a stateful object and has a set of states associated with it. Using the Subscription API, the user can initiate subscriptions, react to incoming subscriptions, refresh subscriptions and ask to terminate subscriptions.

Functions to set and access the subscription fields are also available in the Subscription API.

The Subscription Control functions are:

- `RvSipSubsInit()`
- `RvSipSubsInitStr()`
- `RvSipSubsDialogInit()`
- `RvSipSubsDialogInitStr()`
- `RvSipSubsSubscribe()`
- `RvSipSubsRefresh()`
- `RvSipSubsUnsubscribe()`
- `RvSipSubsRespondAccept()`
- `RvSipSubsRespondPending()`
- `RvSipSubsRespondProvisional()`
- `RvSipSubsRespondReject()`
- `RvSipSubsAutoRefresh()`
- `RvSipSubsSetAlertTimer()`
- `RvSipSubsSetNoNotifyTimer()`
- `RvSipSubsTerminate()`
- `RvSipSubsDetachOwner()`
- `RvSipSubsUpdateSubscriptionTimer()`
RvSipSubsInit()

DESCRIPTION

Initiates a subscription with the following mandatory parameters:

- The To and From headers of the dialog
- The expiresVal of the subscription
- The Event header of the subscription

If the subscription was created within a call-leg you should not set the To and From headers. This function initializes the subscription, but does not send a request. You should call RvSipSubsSubscribe() to send a SUBSCRIBE request.

SYNTAX

RvStatus RvSipSubsInit(
    IN RvSipSubsHandle hSubs,
    IN RvSipPartyHeaderHandle hFrom,
    IN RvSipPartyHeaderHandle hTo,
    IN RvInt32 expiresVal,
    IN RvSipEventHeaderHandle hEvent);

PARAMETERS

hSubs

The handle to the subscription that the user wishes to initialize.

hFrom

The handle to the Party header that contains the From header information.

hTo

The handle to the Party header that contains the To header information.
expiresVal

The expires value, in seconds, to be set in the first SUBSCRIBE request. The value of this parameter must be smaller than 0 x ffffffff/1000. (This is not necessarily the final expires value. The notifier may set a shorter expires value in the 2xx response.) If UNDEFINED, an Expires header will not be set in the SUBSCRIBE request.

hEvent

The handle to the Event header. This header identifies the subscription.

RETURN VALUES

Returns RvStatus.
RvSipSubsInitStr()

DESCRIPTION

Initiates a subscription with the following mandatory parameters in string format:

- The To and From header values of the dialog
- The expiresVal of the subscription
- The Event header value of the subscription

If the subscription was created within a call-leg you should not set the To and From headers. This function initializes the subscription, but does not send a request. You should call RvSipSubsSubscribe() to send a SUBSCRIBE request.

SYNTAX

RvStatus RvSipSubsInitStr(
    IN RvSipSubsHandle hSubs,
    IN RvChar* strFrom,
    IN RvChar* strTo,
    IN RvInt32 expiresVal,
    IN RvChar* strEvent);

PARAMETERS

hSubs

The handle to the subscription that the user wishes to initialize.

strFrom

The string of the From header value. For example, “A<sip:176.56.23.4:4444>”.

strTo

The string of the To header value. For example, “B<sip:176.56.23.4;transport=tcp>”.
expiresVal

The *expires* value, in seconds, to be set in the first SUBSCRIBE request. The value of this parameter must be smaller than 0 x ffffffff/1000. (This is not necessarily the final *expires* value. The notifier may set a shorter *expires* value in the 2xx response.) If UNDEFINED, no Expires header will be set in the SUBSCRIBE request.

strEvent

The string to the Event header value. This header identifies the *subscription*. For example, “eventPackageX.eventTemplateY”. If UNDEFINED, no Expires header will be set in the SUBSCRIBE request.

**RETURN VALUES**

Returns RvStatus.
**RvSipSubsDialogInit()**

**DESCRIPTION**

Initiates a *subscription* with the following dialog parameters:

- To header
- From header
- Remote contact header
- Local contact header

This function is relevant only for a *subscription* that was created outside of a *call-leg*. This function initializes the dialog of the *subscription*, but does not initialize the *subscription* parameters. You should call `RvSipSubsInit()` or `RvSipSubsReferInit()` to initialize the *subscription* parameters.

**SYNTAX**

```c
RvStatus RvSipSubsDialogInit(
    IN RvSipSubsHandle hSubs,
    IN RvSipPartyHeaderHandle hFrom,
    IN RvSipPartyHeaderHandle hTo,
    IN RvSipAddressHandle hLocalContactAddr,
    IN RvSipAddressHandle hRemoteContactAddr);
```

**PARAMETERS**

- **hSubs**
  The handle to the *subscription* that the user wishes to initialize.

- **hFrom**
  The handle to a party header, containing the From header information.

- **hTo**
  The handle to a party header containing the To header information.

- **hLocalContactAddr**
  The handle to address containing the local contact address information.
Subscription Control API
RvSipSubsDialogInit()

`hRemoteContactAddr`

The handle to address containing the remote contact address information.

**RETURN VALUES**

Returns `RvStatus`. 
RvSipSubsDialogInitStr()

DESCRIPTION

Initiates a subscription with the following dialog parameters in string format:
- To header
- From header
- Remote contact header
- Local contact header

This function is relevant only for a subscription that was created outside of a call-leg. This function initializes the dialog of the subscription, but does not initialize the subscription parameters. You should call RvSipSubsInitStr() or RvSipSubsReferInitStr() to initialize the subscription parameters.

SYNTAX

RvStatus RvSipSubsDialogInitStr(
    IN RvSipSubsHandle hSubs,
    IN RvChar* strFrom,
    IN RvChar* strTo,
    IN RvChar* strLocalContactAddr,
    IN RvChar* strRemoteContactAddr);

PARAMETERS

hSubs
The handle to the subscription the user wishes to initialize.

strFrom
The string of the From header value. For example, “A<sip:176.56.23.4:4444>”.

strTo
The string of the To header value. For example, “B<sip:176.56.23.4;transport=TCP>”.

Subscription Control API
RvSipSubsDialogInitStr()

strLocalRemote
The string of the local contact address value. For example, “sip:176.56.23.4:4444”.

strRemoteContact
String of the remote contact address value. For Example: "sip:176.56.23.4:4444"

RETURN VALUES
Returns RvStatus.
RvSipSubsSubscribe()

DESCRIPTION
Establishes a subscription by sending a SUBSCRIBE request. This function may be called only after the subscription was initialized with the To, From, Event and Expires parameters. (If this is a subscription of an already established call-leg, the To and From parameters are taken from the call-leg). Calling this function causes a SUBSCRIBE request to be sent and the Subscription state machine to progress to the SUBS_SENT state.

SYNTAX
RvStatus RvSipSubsSubscribe(
    IN RvSipSubsHandle hSubs);

PARAMETERS
hSubs
The handle to the subscription that the user wishes to subscribe.

RETURN VALUES
Returns RvStatus.
RvSipSubsRefresh()

DESCRIPTION

Refreshes a subscription by sending a SUBSCRIBE request in the subscription. This method can be called only in the SUBS_ACTIVE state. Calling the refresh function causes a SUBSCRIBE request to be sent, and the Subscription state machine to advance to the SUBS_REFRESHING state.

SYNTAX

RvStatus RvSipSubsRefresh(
    IN RvSipSubsHandle hSubs,
    IN RvInt32 refreshExpiresVal);

PARAMETERS

hSubs

The handle to the subscription that the user wishes to refresh.

refreshExpiresVal

The expires value, in seconds, to be set in the REFRESH request. The value of this parameter must be smaller than 0x fffffff/1000.

RETURN VALUES

Returns RvStatus.
RvSipSubsUnsubscribe()

DESCRIPTION
Asks the notifier to terminate the subscription by sending a SUBSCRIBE request with an “Expires:0” header in the subscription. This function can be called in the SUBS_ACTIVE, SUBS_PENDING and SUBS_2XX RCVD states. Calling this function advances the Subscription state machine to the SUBS_UNSUBSCRIBING state.

SYNTAX
RvStatus RvSipSubsUnsubscribe(
    IN RvSipSubsHandle hSubs);

PARAMETERS
hSubs
The handle to the subscription that the user wishes to unsubscribe.

RETURN VALUES
Returns RvStatus.
RvSipSubsRespondAccept()

DESCRIPTION
Sends a 200 response on a SUBSCRIBE request and updates the Subscription state machine. You can use this function to accept an initial SUBSCRIBE, Refresh-SUBSCRIBE or UNSUBSCRIBE request.

Note This function does not send a NOTIFY request. It is the responsibility of the application to create and send a NOTIFY request with an “active” Subscription-State header value immediately after the application accepts the SUBSCRIBE request by calling this function.

SYNTAX
RvStatus RvSipSubsRespondAccept(
    IN RvSipSubsHandle hSubs,
    IN RvInt32 expiresVal);

PARAMETERS

hSubs
The handle to the subscription that the application wishes to accept.

expiresVal
The expires value, in seconds, to be set in the 200 response. If UNDEFINED, the expires value will be the same as in the SUBSCRIBE request. The value of this parameter must be smaller than 0 x ffffffff/1000.

Note If there is no Expires header in the incoming SUBSCRIBE request, you must not give an UNDEFINED expires value.

RETURN VALUES
Returns RvStatus.
RvSipSubsRespondPending()

DESCRIPTION

Sends a 202 response for an initial SUBSCRIBE request, and updates the Subscription state machine to the SUBS_PENDING state. You may use this function only for an initial subscribe request.

**Note** This function does not send a NOTIFY request. It is the responsibility of the application to create and send a NOTIFY request with “pending” as the Subscription-State header value immediately after the application responds with this function to the SUBSCRIBE request.

SYNTAX

```c
RvStatus RvSipSubsRespondPending(
    IN RvSipSubsHandle hSubs,
    IN RvInt32 expiresVal);
```

PARAMETERS

**hSubs**

The handle to the subscription.

**expiresVal**

The expires value, in seconds, to be set in the 202 response. If UNDEFINED, the expires value will be the same as in the SUBSCRIBE request. The value of this parameter must be smaller than 0 x ffffffff/1000.

**Note** If there is no Expires header in the incoming SUBSCRIBE request, you must not give an UNDEFINED expires value.

RETURN VALUES

Returns RvStatus.
RvSipSubsRespondProvisional()

DESCRIPTION
Sends a 1xx on a SUBSCRIBE request (initial-SUBSCRIBE, refresh-SUBSCRIBE or UNSUBSCRIBE). This function can be used in the following states:

- RVSIP_SUBS_STATE_SUBS_RCVD
- RVSIP_SUBS_STATE_REFRESH_RCVD
- RVSIP_SUBS_STATE_UNSUBSCRIBE_RCVD

Note  According to RFC 3265, sending a provisional response on a SUBSCRIBE request is not allowed. According to RFC 3515, only a 100 response on a REFER request is allowed.

SYNTAX

RvStatus RvSipSubsRespondProvisional(
    IN RvSipSubsHandle hSubs,
    IN RvUint16 statusCode,
    IN RvChar* strReasonPhrase)

PARAMETERS

hSubs
The handle to the subscription for which the user wishes to send the 1xx.

statusCode
The value of 1xx.

strReasonPhrase
The reason phrase to be set in the sent message. (Can be NULL for setting a default reason phrase.)

RETURN VALUES
Returns RvStatus.
Subscribe Control API
RvSipSubsRespondReject()

RvSipSubsRespondReject()

DESCRIPTION
Rejects an incoming SUBSCRIBE request (initial SUBSCRIBE, REFRESH or UNSUBSCRIBE). This function can be used in the following states:
- RVSIP_SUBS_STATE_SUBS_RCVD
- RVSIP_SUBS_STATE_REFRESH_RCVD
- RVSIP_SUBS_STATE_UNSUBSCRIBE_RCVD

SYNTAX
RvStatus RvSipSubsRespondReject(
    IN RvSipSubsHandle hSubs,
    IN RvUint16 statusCode,
    IN RvChar* strReasonPhrase);

PARAMETERS

hSubs
The handle to the subscription that the application wishes to reject.

statusCode
The rejection response code.

strReasonPhrase
The reason phrase to be set in the reject message that is sent. (Can be NULL to set a default reason phrase.)

RETURN VALUES
Returns RvStatus.
RvSipSubsAutoRefresh()

DESCRIPTION
Sets the “autoRefresh” flag of a specific subscription. When bAutoRefresh is set to RV_TRUE, an automatic refreshing SUBSCRIBE request is sent when needed. The default value of the bAutoRefresh parameter is taken from the configuration.

SYNTAX
RvStatus RvSipSubsAutoRefresh(
    IN RvSipSubsHandle hSubs,
    IN RvBool bAutoRefresh);

PARAMETERS
hSubs
The handle to the subscription.

bAutoRefresh
RV_TRUE to enable auto refresh; RV_FALSE to disable auto refresh.

RETURN VALUES
Returns RvStatus.
Subscription Control API
RvSipSubsSetAlertTimer()

RvSipSubsSetAlertTimer()

DESCRIPTION
Sets the alert timer value of a specific subscription. The alert timer value defines when an alert is given before the expiration of a subscription. The default value of alertValue is taken from the configuration.

SYNTAX
RvStatus RvSipSubsSetAlertTimer(
    IN RvSipSubsHandle hSubs,
    IN RvInt32 alertValue);

PARAMETERS

hSubs
The handle to the subscription.

alertValue
The value in milliseconds.

RETURN VALUES
Returns RvStatus.
RvSipSubsSetNoNotifyTimer()

DESCRIPTION

Sets the no-notify timer value of a specific subscription. The no-notify timer value defines when the subscription is terminated, if no NOTIFY request is received and accepted after a 2xx on a SUBSCRIBE request was received. The default value of this parameter is taken from configuration.

SYNTAX

RvStatus RvSipSubsSetNoNotifyTimer(
    IN RvSipSubsHandle hSubs,
    IN RvInt32 noNotifyValue);

PARAMETERS

hSubs

The handle to the subscription.

noNotifyValue

The value in milliseconds.

RETURN VALUES

Returns RvStatus.
RvSipSubsTerminate()

DESCRIPTION
Terminates a subscription without sending any messages—UNSUBSCRIBE or NOTIFY (with “terminated” as the Subscription-State header value). The subscription assumes the TERMINATED state.

Calling this function causes an abnormal termination. All notifications and transactions that are related to the subscription are also terminated. If the subscription was created with no related call-leg, the subscription dialog is also terminated.

SYNTAX
RvStatus RvSipSubsTerminate(
    IN RvSipSubsHandle hSubs);

PARAMETERS
hSubs
The handle to the subscription that the user wishes to terminate.

RETURN VALUES
Returns RvStatus.
RvSipSubsDetachOwner()

DESCRIPTION
Detaches the subscription from its owner. After calling this function the application will stop receiving events for this subscription. This function can be called only when the object is in the TERMINATED state.

SYNTAX
RvStatus RvSipSubsDetachOwner(
    IN RvSipSubsHandle hSubs);

PARAMETERS
hSubs
The handle to the subscription the user wishes to detach from its owner.

RETURN VALUES
Returns RvStatus.
RvSipSubsUpdateSubscriptionTimer()

DESCRIPTION
Sets the subscription timer. When the RvSipSubsSubscriptionExpiredEv() callback function is called, the subscription timer is no longer active. Calling this function re-activates this timer. (On the next expiration, RvSipSubsSubscriptionExpiredEv() will be called again).

SYNTAX
RvStatus RvSipSubsUpdateSubscriptionTimer(
    IN RvSipSubsHandle hSubs,
    IN RvInt32 expirationValue
)

PARAMETERS

hSubs
The handle to the subscription.

expirationValue
The expiration value, in mili-seconds.

RETURN VALUES
Returns RvStatus.
The Subscription Get and Set functions are:

- RvSipSubsIsOutOfBand()
- RvSipSubsSetEventHeader()
- RvSipSubsGetEventHeader()
- RvSipSubsSetExpiresVal()
- RvSipSubsGetExpiresVal()
- RvSipSubsGetRequestedExpiresVal()
- RvSipSubsGetDialogObj()
- RvSipSubsSetAppSubsHandle()
- RvSipSubsGetSubsMgr()
- RvSipSubsGetCurrentState()
- RvSipSubsGetSubsType()
- RvSipSubsGetReceivedMsg()
- RvSipSubsGetOutboundMsg()
- RvSipSubsGetStackInstance()
- RvSipSubsGetEventPackageType()
- RvSipSubsGetNewHeaderHandle()
- RvSipSubsSetForkingEnabledFlag()
- RvSipSubsGetForkingEnabledFlag()
- RvSipSubsGetOriginalSubs()
- RvSipSubsGetTranscCurrentLocalAddress()
- RvSipSubsSetRejectStatusCodeOnCreation()
RvSipSubsIsOutOfBand()

DESCRIPTION
Indicates whether or not this is an out-of-band subscription.

SYNTAX
RvStatus RvSipSubsIsOutOfBand(
    IN RvSipSubsHandle hSubs,
    OUT RvBool* bIsOutOfBand);

PARAMETERS
hSubs
The handle to the subscription.

bIsOutOfBand
RV_TRUE if the subscription is out-of-band. Otherwise, RV_FALSE.

RETURN VALUES
Returns RvStatus.
**RvSipSubsSetEventHeader()**

**DESCRIPTION**

Sets the Event header associated with the subscription.

**Note** The Event header identifies the subscription so you must not change it after subscription was initialized.

**SYNTAX**

```c
RvStatus RvSipSubsSetEventHeader(
    IN RvSipSubsHandle hSubs,
    IN RvSipEventHeaderHandle hEvent);
```

**PARAMETERS**

- **hSubs**
  The handle to the subscription.

- **hEvent**
  The handle to an application-constructed Event header.

**RETURN VALUES**

Returns RvStatus.
Subscription Get and Set API

RvSipSubsGetEventHeader()

**DESCRIPTION**

Returns the Event header associated with the *subscription*.

**Note**  The Event header identifies the *subscription* so you must not change it after *subscription* was initialized.

**SYNTAX**

```
RvStatus RvSipSubsGetEventHeader(
    IN   RvSipSubsHandle     hSubs,
    OUT  RvSipEventHeaderHandle  *phEvent);
```

**PARAMETERS**

- **hSubs**
  The handle to the *subscription*.

- **phEvent**
  A pointer to the *subscription* Event header handle.

**RETURN VALUES**

Returns **RvStatus**.
**RvSipSubsSetExpiresVal()**

**DESCRIPTION**
Sets the Expires header value, in seconds, associated with the subscription. You can only set the expires value before sending the first SUBSCRIBE request. Afterwards, you can change the value only with the RvSipSubsRefresh() function. This is because the expires value is set according to the 2xx response or NOTIFY messages.

You must not change the value after the subscription was initialized.

**SYNTAX**

```c
RvStatus RvSipSubsSetExpiresVal(
    IN RvSipSubsHandle hSubs,
    IN RvInt32 expiresVal);
```

**PARAMETERS**

- **hSubs**
  The handle to the subscription.

- **expiresVal**
  The time, in seconds, for the subscription expiration. The value of this parameter must be smaller than 0x ffffffff/10000.

**RETURN VALUES**

Returns RvStatus.
RvSipSubsGetExpiresVal()

DESCRIPTION
Returns the expires value of the subscription, in seconds. For a notifier subscription, this function retrieves the last expires value that was set by the notifier in a 2xx response, or in a NOTIFY request. (If the SUBSCRIBE request did not contain an Expires header, RVSIP_SUBS_EXPIRES_VAL_LIMIT is retrieved.)

If the notifier wants the expires value from the incoming SUBSCRIBE request it should use RvSipSubsGetRequestedExpiresVal() function. For a subscriber subscription, this function retrieves the value that was set on initialization of the subscription, or the updated value if the notifier already updated it.

SYNTAX
RvStatus RvSipSubsGetExpiresVal(
    IN RvSipSubsHandle hSubs,
    OUT RvInt32 *pExpires);

PARAMETERS

hSubs
The handle to the subscription.

pExpires
A pointer to the expires value.

RETURN VALUES
Returns RvStatus.
RvSipSubsGetRequestedExpiresVal()

DESCRIPTION

Returns the requested expires value, in seconds. When a new SUBSCRIBE request is received, its expires value can be retrieved using this function. (If the SUBSCRIBE request did not contain an Expires header, RVSIP_SUBS_EXPIRES_VAL_LIMIT is retrieved.) This function is relevant only for a notifier subscription.

Note  This function is different from RvSipSubsGetExpiresVal() because it only retrieves a value when there is an incoming SUBSCRIBE Request that is waiting for a response. RvSipSubsGetExpiresVal() retrieves the last expires value that was set by a notifier in a 2xx response, or in a NOTIFY request.

SYNTAX

RvStatus RvSipSubsGetRequestedExpiresVal(IN RvSipSubsHandle hSubs, OUT RvInt32 *pRequestedExpires);

PARAMETERS

hSubs

The handle to the subscription.

pRequestedExpires

A pointer to the requested expires value.

RETURN VALUES

Returns RvStatus.
Subscription Get and Set API
RvSipSubsGetDialogObj()

**RvSipSubsGetDialogObj()**

**DESCRIPTION**

Returns the handle to the dialog object that related to this subscription. (This function is relevant for a subscription that was not created in a connected call-leg.) The user can use this dialog handle to set or get parameters that are kept inside the dialog, and not within the subscription (such as Call-ID, cseq, To, From and OutboundAddress). You can set and get dialog parameters with the Call-leg API functions.

**Note**  You must not change the dialog parameters after the subscription is established.

**SYNTAX**

```c
RvStatus RvSipSubsGetDialogObj(
    IN  RvSipSubsHandle  hSubs,
    OUT RvSipCallLegHandle  *phDialog);
```

**PARAMETERS**

**hSubs**

The handle to the subscription.

**phDialog**

A pointer to the dialog handle.

**RETURN VALUES**

Returns RvStatus.
**RvSipSubsSetAppSubsHandle()**

**DESCRIPTION**
Sets the application subscription handle.

**SYNTAX**

```c
RvStatus RvSipSubsSetAppSubsHandle(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs);
```

**PARAMETERS**

- **hSubs**
  The handle to the subscription.

- **hAppSubs**
  The handle to the application subscription.

**RETURN VALUES**

Returns `RvStatus`.
RvSipSubsGetAppSubsHandle()

DESCRIPTION
Returns the application subscription handle.

SYNTAX
RvStatus RvSipSubsGetAppSubsHandle(
    IN  RvSipSubsHandle  hSubs,
    OUT RvSipAppSubsHandle *phAppSubs);

PARAMETERS
hSubs
The handle to the subscription.

hAppSubs
The handle to the application subscription.

RETURN VALUES
Returns RvStatus.
RvSipSubsGetSubsMgr()

DESCRIPTION

Returns the SubscriptionMgr handle and the application SubscriptionMgr handle to the given subscription.

SYNTAX

RvStatus RvSipSubsGetSubsMgr(
    IN RvSipSubsHandle hSubs,
    OUT RvSipSubsMgrHandle *phSubsMgr,
    OUT void **phSubsAppMgr);

PARAMETERS

hSubs

The handle to the subscription.

phSubsMgr

The SubscriptionMgr handle.

phSubsAppMgr

The application SubscriptionMgr handle.

RETURN VALUES

Returns RvStatus.
Subscription Get and Set API
RvSipSubsGetCurrentState()

**RvSipSubsGetCurrentState()**

**DESCRIPTION**

Gets the current state of the subscription.

**SYNTAX**

```c
RvStatus RvSipSubsGetCurrentState(
    IN RvSipSubsHandle hSubs,
    OUT RvSipSubsState *peState);
```

**PARAMETERS**

- **hSubs**
  The handle to the subscription.

- **peState**
  The current state of the subscription.

**RETURN VALUES**

Returns RvStatus.
RvSipSubsGetSubsType()

DESCRIPTION
Gets the subscription type. A subscription can be either a subscriber or a notifier.

SYNTAX
RvStatus RvSipSubsGetSubsType(
    IN RvSipSubsHandle hSubs,
    OUT RvSipSubscriptionType *peSubsType);

PARAMETERS
hSubs
The handle to the subscription.

peSubsType
The subscription type.

RETURN VALUES
Returns RvStatus.
**RvSipSubsGetReceivedMsg()**

**DESCRIPTION**
Gets the message that was received by the subscription. You can call this function from the state changed callback function when the new state indicates that a message was received. If there is no valid received message, NULL will be returned.

**SYNTAX**

```c
RvStatus RvSipSubsGetReceivedMsg(
    IN RvSipSubsHandle hSubs,
    OUT RvSipMsgHandle *phMsg;
)
```

**PARAMETERS**

- **hSubs**
  The handle to the subscription.

- **phMsg**
  A pointer to the received message.

**RETURN VALUES**
Returns RvStatus.
RvSipSubsGetOutboundMsg()

DESCRIPTION

Gets the message (request or response) that will be sent by the subscription. You can get the message only in states in which the subscription may send a message. You can call this function before you call control API functions that send a message (such as RvSipSubsSubscribe()), or from the expirationAlert callback function when a REFRESH message is going to be sent.

Note  The outbound message you receive 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 RvSipSubsMsgToSendEv().

SYNTAX

RvStatus RvSipSubsGetOutboundMsg(
    IN RvSipSubsHandle hSubs,
    OUT RvSipMsgHandle *phMsg;
)

PARAMETERS

hSubs
The subscription handle.

phMsg
A pointer to the message.

RETURN VALUES

Returns RvStatus.
RvSipSubsGetStackInstance()

DESCRIPTION
Returns the handle to the Stack instance to which this subscription belongs.

SYNTAX
RvStatus RvSipSubsGetStackInstance(
    IN RvSipSubsHandle hSubs,
    OUT void **phStackInstance);

PARAMETERS
hSubs
The handle to the subscription.

phStackInstance
A valid pointer which will be updated with a handle to the Stack instance.

RETURN VALUES
Returns RvStatus.
RvSipSubsGetEventPackageType()

DESCRIPTION

Returns the event package type of the given subscription: refer, presence, presence-winfo, and so on. Note that in the case of presence-winfo, the Event header contains a “package: presence”, and a “template: winfo”, which are both combined into a single subscription event package, PRESENCE_WINFO.

SYNTAX

RvStatus RvSipSubsGetEventPackageType(
    IN RvSipSubsHandle hSubs,
    OUT RvSipSubsEventPackageType* pePackageType);

PARAMETERS

hSubs
The handle to the subscription.

pePackageType
A pointer to the event package type.

RETURN VALUES

Returns RvStatus.
**RvSipSubsGetNewHeaderHandle()**

**DESCRIPTION**

Allocates a new header on the *subscription* page, and returns the new header handle. The application may use this function to allocate a *message* header. It should then fill the element information and set it back to the *subscription* using the relevant Set (or “Init”) function. (For example, after getting a new event header handle, the application should fill it, and then set it back with **RvSipSubsSetEventHeader()**. The function supports the following elements:

- **To/From**—you should set these headers back with **RvSipSubsInit()** or **RvSipSubsDialogInit()**.
- **Event**—you should set this header back with **RvSipSubsSetEventHeader()**.
- **Refer-to**—you should set this header back with **RvSipSubsReferInit()**.
- **Referred-by**—you should set this header back with **RvSipSubsReferInit()**.

**SYNTAX**

```c
RvStatus RvSipSubsGetNewHeaderHandle(
    IN RvSipSubsHandle hSubs,
    IN RvSipHeaderType eHeaderType,
    OUT void*  phHeader);
```

**PARAMETERS**

- **hSubs**
  The handle to the *subscription*.

- **eHeaderType**
  The type of header to allocate.

- **phHeader**
  A pointer to the new header handle.
Subscription Get and Set API
RvSipSubsGetNewHeaderHandle()

RETURN VALUES
Returns RvStatus.
RvSipSubsSetForkingEnabledFlag()

DESCRIPTION
Sets the “forking enabled” flag into a subscription. In case the SUBSCRIBE request was forked by a proxy to several notifiers, the subscription may get NOTIFY requests from several notifiers. Setting the “forking enabled” flag of a subscription to RV_TRUE will create a new subscription for each forked-notify that was received for the original subscription (the subscription that sent the SUBSCRIBE request).
Setting the “forking enabled” flag of a subscription to RV_FALSE will cause the original subscription to reject with a 481 response (No Matching Call) for every forked-notify that is received. For forking details, see RFC 3265, sections 3.3.3 and 4.4.9.

SYNTAX
RvStatus RvSipSubsSetForkingEnabledFlag(
    IN RvSipSubsHandle hSubs,
    IN RvBool bForkingEnabled);

PARAMETERS
hSubs
The handle to the subscription that sent the request.

bForkingEnabled
The value of the “forking enabled” flag.

RETURN VALUES
Returns RvStatus.
RvSipSubsGetForkingEnabledFlag()

DESCRIPTION

Gets the “forking enabled” flag from a subscription. In case the SUBSCRIBE request was forked by a proxy to several notifiers, the subscription may get NOTIFY requests from several notifiers. Setting the “forking enabled” flag of a subscription to RV_TRUE will create a new subscription for each forked-notify that was received for the original subscription (the subscription that sent the SUBSCRIBE request).

Setting the “forking enabled” flag of a subscription to RV_FALSE will cause the original subscription to reject with a 481 response (No Matching Call) for every forked-notify that is received. For forking details, see RFC 3265, sections 3.3.3 and 4.4.9.

SYNTAX

RvStatus RvSipSubsGetForkingEnabledFlag(
    IN RvSipSubsHandle hSubs,
    OUT RvBool *pbForkingEnabled);

PARAMETERS

hSubs

The handle to the subscription that sent the request.

pbForkingEnabled

A valid pointer which will be updated with the subscription “forking enabled” Boolean.

RETURN VALUES

Returns RvStatus.
**RvSipSubsGetOriginalSubs()**

**DESCRIPTION**

Gets handle to the original subscription. In case a SUBSCRIBE request was forked by a proxy to several notifiers, the subscription may get NOTIFY requests from several notifiers. The original subscription is the subscription that actually sent the SUBSCRIBE request. If no original subscription exists, NULL will be retrieved. This function is for use with forked subscriptions. For forking details, see RFC 3265, sections 3.3.3 and 4.4.9.

**SYNTAX**

```
RvStatus RvSipSubsGetOriginalSubs(
    IN RvSipSubsHandle hSubs,
    OUT RvSipSubsHandle *phOriginalSubs);
```

**PARAMETERS**

- **hSubs**
  
  The handle to the subscription that sent the request.

- **phOriginalSubs**
  
  A valid pointer which will be updated with the original subscription handle.

**RETURN VALUES**

Returns RvStatus.
**RvSipSubsGetTranscCurrentLocalAddress()**

**DESCRIPTION**

Gets the local address that is used by a specific *subscription transaction*. You supply a *subscription* handle to get the local address of a Subscribe *transaction*, or a *notification* handle to get the local address of a NOTIFY *transaction*.

**SYNTAX**

```c
RvStatus RvSipSubsGetTranscCurrentLocalAddress(
    IN RvSipSubsHandle hSubs,
    IN RvSipNotifyHandle hNotify,
    OUT RvSipTransport *eTransportType,
    OUT RvSipTransportAddressType *eAddressType,
    OUT RvChar *localAddress,
    OUT RvUint16 *localPort);
```

**PARAMETERS**

**hSubs**

The handle to the *subscription* that is holding the *transaction*.

**hNotify**

The handle to the *notification* that is holding the *transaction*.

**eTransportType**

The transport type (UDP, TCP or TLS).

**eAddressType**

The address type (IP4 or IP6).

**localAddress**

The local address this *transaction* is using. (The value must be longer than 48 bytes.)
Subscription Get and Set API
RvSipSubsGetTranscCurrentLocalAddress()

**localPort**
The local port that this *transaction* is using.

**RETURN VALUES**
Returns *RvStatus.*
**RvSipSubsSetRejectStatusCodeOnCreation()**

**DESCRIPTION**

This function can be used synchronously from the `RvSipSubsCreatedEv()` callback to instruct the SIP Stack to automatically reject the request that created this *subscription*. You should supply the reject status code in this function. If you set this status code, the *subscription* will be destructed automatically when the `RvSipSubsCreatedEv()` returns. The application will not get any further callbacks that relate to this *subscription*. The application will not get the `RvSipSubsMsgToSendEv()` for the reject response message or the TERMINATED state for the *subscription*.

This function should not be used for rejecting a request in a normal scenario. For this, you should use the `RvSipSubsRespondReject()` function. You should use this function only if your application is incapable of handling this new *subscription* at all, for example, in a situation where an application is out of resources.

**Note**  When this function is used to reject a request, the application cannot use the outbound message mechanism to add information to the outgoing response message. If you wish to change the response message, you must use the regular reject function in the RVSIP_SUBS_STATE_SUBS_RCVD state.

**SYNTAX**

```c
RvStatus RvSipSubsSetRejectStatusCodeOnCreation(
    IN RvSipSubsHandle hSubs,
    IN RvUint16 rejectStatusCode);
```

**PARAMETERS**

- **hSubs**
  
  The handle to the *subscription*.

- **rejectStatusCode**
  
  The reject status code for rejecting the request that created this object. The value must be from 300 to 699.
Subscription Get and Set API

RETURN VALUES

Returns RvStatus.
A notification represents a SIP notification as defined in draft sip-event-05. Using the Notification API, the user can create notifications and react to incoming notifications. Functions to set and access the notification fields are also available in the Notification API.

The Subscription Notification functions are:

- `RvSipSubsCreateNotify()`
- `RvSipNotifySend()`
- `RvSipNotifyAccept()`
- `RvSipNotifyReject()`
- `RvSipNotifyTerminate()`
- `RvSipNotifyDetachOwner()`
- `RvSipNotifyGetOutboundMsg()`
- `RvSipNotifyGetStackInstance()`
- `RvSipNotifyGetSubsState()`
- `RvSipNotifySetSubscriptionStateParams()`
RvSipSubsCreateNotify()

DESCRIPTION

Creates a new notification that is related to a given subscription and exchange handles with the application notification. For setting notify state information in the NOTIFY request before sending, use the RvSipNotifyGetOutboundMsg() function.

SYNTAX

RvStatus RvSipSubsCreateNotify(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppNotifyHandle hAppNotify,
    OUT RvSipNotifyHandle *phNotify)

PARAMETERS

hSubs

The handle to the subscription that relates to the new notification.

hAppNotify

The handle to the application notification.

phNotify

The handle to the newly created notification.

RETURN VALUES

Returns RvStatus.
RvSipNotifySend()

**DESCRIPTION**

Sends the NOTIFY message placed in the *notification*.

**SYNTAX**

```c
RvStatus RvSipNotifySend(
    IN RvSipNotifyHandle hNotify);
```

**PARAMETERS**

- **hNotify**
  
The handle to the *notification*.

**RETURN VALUES**

Returns *RvStatus*. 
Subscription Notification API

RvSipNotifyAccept()

**DESCRIPTION**

Sends a 200 response on a NOTIFY request and destructs the notification. This function influences the Subscription state machine as follows:

- If the subscription is in the SUBS_SENT state and accepts a NOTIFY (active or pending) request, changes to the SUBS_NOTIFY_BEFORE_2XX_RCVD state.
- If the subscription is in the SUBS_2XX_RCVD state and accepts a NOTIFY (pending) request, changes to the SUBS_PENDING state.
- If the subscription is in the SUBS_2XX_RCVD state and accepts a NOTIFY (active) request, changes to the SUBS_ACTIVE state.
- If the subscription is in the SUBS_PENDING state and accepts a NOTIFY (active) request, changes to the SUBS_ACTIVE state.
- If the subscription is in all states and accepts a NOTIFY (terminated) request, changes to the SUBS_TERMINATED state.

**SYNTAX**

```
RvStatus RvSipNotifyAccept(
    IN RvSipNotifyHandle   hNotify);
```

**PARAMETERS**

- **hNotify**
  
  The handle to the notification.

**RETURN VALUES**

Returns RvStatus.
RvSipNotifyReject()

DESCRIPTION
Sends a non-2xx response on a NOTIFY request, and destructs the *notification*. This function does not influence the Subscription state machine.

SYNTAX
RvStatus RvSipNotifyReject(
    IN RvSipNotifyHandle hNotify,
    IN RvUint16 statusCode);

PARAMETERS
hNotify
The handle to the *notification*.

statusCode
The response code.

RETURN VALUES
Returns RvStatus.
RvSipNotifyTerminate()

DESCRIPTION
Terminates a notification without sending any messages. The notification will inform of the terminated status. Calling this function will cause abnormal termination.

SYNTAX
RvStatus RvSipNotifyTerminate(
    IN RvSipNotifyHandle hNotify);

PARAMETERS
hNotify
The handle to the notification.

RETURN VALUES
Returns RvStatus.
**RvSipNotifyDetachOwner()**

**DESCRIPTION**
Detaches the NOTIFY object from its owner. After calling this function the user will stop receiving events for this NOTIFY object. This function can be called only when the object is in the TERMINATED state.

**SYNTAX**
```
RvStatus RvSipNotifyDetachOwner(
    IN RvSipNotifyHandle hNotify);
```

**PARAMETERS**
- **hNotify**
The handle to the notification.

**RETURN VALUES**
Returns `RvStatus`.
Subscription Notification API

RvSipNotifyGetOutboundMsg()

**DESCRIPTION**

Gets the message (request or response) that is going to be sent by the notification. You can get the message only in the states in which the notification may send a message. You can call this function before you call a control API functions that send a message, such as RvSipNotifySend() or RvSipNotifyAccept().

**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 and a body to the message before it is sent. To view the complete message, use RvSipSubsMsgToSendEv().

**SYNTAX**

```c
RvStatus RvSipNotifyGetOutboundMsg(
    IN RvSipNotifyHandle hNotify,
    OUT RvSipMsgHandle *phMsg);
```

**PARAMETERS**

- **hNotify**
  The notification handle.

- **phMsg**
  A pointer to the message.

**RETURN VALUES**

Returns RvStatus.
**RvSipNotifyGetStackInstance()**

**DESCRIPTION**
Returns the handle to the Stack instance to which this *notification* belongs.

**SYNTAX**

```c
RvStatus RvSipNotifyGetStackInstance(
    IN RvSipNotifyHandle hNotify,
    OUT void **phStackInstance);
```

**PARAMETERS**

- **hNotify**
The *notification* handle.

- **phStackInstance**
A valid pointer which will be updated with a handle to the Stack instance.

**RETURN VALUES**
Returns `RvStatus`. 
RvSipNotifyGetSubsState()

DESCRIPTION
Returns the value of the Subscription-State header that was set by application to the NOTIFY message of this notify object.

SYNTAX
RvStatus RvSipNotifyGetSubsState (  
    IN RvSipNotifyHandle hNotify,  
    OUT RvSipSubscriptionSubstate* peSubsState);  

PARAMETERS
hNotify
The handle to the notification.

peSubsState
The state of the subscription—active, pending, terminated or other.

RETURN VALUES
Returns RvStatus.
**RvSipNotifySetSubscriptionStateParams()**

**DESCRIPTION**

Creates and sets the Subscription-State header in the *notification*. This header will be set to the outbound NOTIFY request.

**SYNTAX**

```c
RvStatus RvSipNotifySetSubscriptionStateParams(
    IN RvSipNotifyHandle hNotify,
    IN RvSipSubscriptionSubstate eSubsState,
    IN RvSipSubscriptionReason eReason,
    IN RvInt32 expiresParamVal);
```

**PARAMETERS**

- **hNotify**
  The handle to the *notification*.

- **eSubsState**
  The SubsState to set in the Subscription-State header.

- **eReason**
  The reason to set in the Subscription-State header
  (RVSIP_SUBSCRIPTION_REASON_UNDEFINED for no reason).

- **expiresParamVal**
  The expires parameter value to set in the Subscription-State header (may be UNDEFINED).

**RETURN VALUES**

Returns *RvStatus*. 
Subscription Authentication API

The Subscription Authentication functions are:

- `RvSipSubsAuthBegin`
- `RvSipSubsAuthProceed`
- `RvSipSubsRespondUnauthenticated`
- `RvSipSubsRespondUnauthenticatedDigest`
RvSipSubsAuthBegin()

DESCRIPTION
Begins the server authentication process by challenging an incoming SUBSCRIBE, REFER or NOTIFY request. If the request is SUBSCRIBE or REFER, the value for the hSubs handle should be given, and hNotify should be NULL. If the request is NOTIFY, hNotify should be given, and hSubs can be NULL.

SYNTAX
RvStatus RvSipSubsAuthBegin(
    IN RvSipSubsHandle hSubs,
    IN RvSipNotifyHandle hNotify);

PARAMETERS
hSubs
The handle to the subscription that the user wishes to challenge.

hNotify
The handle to the notification that the user wishes to challenge.

RETURN VALUES
Returns RvStatus.
Subscription Authentication API
RvSipSubsAuthProceed()

RvSipSubsAuthProceed()

DESCRIPTION

Orders the Stack to proceed with the authentication procedure.

The action options are:

- **RVSIP_TRANSC_AUTH_ACTION_USE_PASSWORD**—checks the given Authorization header with the given password.
- **RVSIP_TRANSC_AUTH_ACTION_SUCCESS**—the user checked the Authorization header and it is correct. (This will cause `RvSipSubsAuthCompletedEv()` to be called, with success status.)
- **RVSIP_TRANSC_AUTH_ACTION_FAILURE**—the user wants to stop the loop that searches for Authorization headers. (This will cause `RvSipSubsAuthCompletedEv()` to be called, with failure status.)
- **RVSIP_TRANSC_AUTH_ACTION_SKIP**—orders to skip the given header, and continue the authentication procedure with next the next header, if it exists. (This will cause `RvSipSubsAuthCredentialsFoundEv()` to be called, or `RvSipSubsAuthCompletedEv()` with failure status, if there are no more Authorization headers.

SYNTAX

```c
RvStatus RvSipSubsAuthProceed(
    IN RvSipSubsHandle hSubs,
    IN RvSipNotifyHandle hNotify,
    IN RvSipTransactionAuthAction action,
    IN RvSipAuthorizationHeaderHandle hAuthorization,
    IN RvChar *password);
```

PARAMETERS

- **hSubs**

  The handle to the `subscription` that the user challenges.
**Subscription Authentication API**

**RvSipSubsAuthProceed()**

**hNotify**
The handle to the *notification* that the user challenges.

**action**
Specifies with which action to proceed (see above).

**hAuthorization**
The handle to the Authorization header for which the function checks authentication. (This parameter is needed if the action is RVSIP_TRANSC_AUTH_ACTION_USE_PASSWORD, otherwise NULL.)

**password**
The password for the realm+userName in the header. (This parameter is needed if the action is RVSIP_TRANSC_AUTH_ACTION_USE_PASSWORD, otherwise, NULL.)

**RETURN VALUES**
Returns RvStatus.
RvSipSubsRespondUnauthenticated()

DESCRIPTION
Sends 401/407 response on a SUBSCRIBE, REFER or NOTIFY request and adds a copy of the given header to the response message. The given header should contain the authentication parameters (Authenticate header or Other header).

SYNTAX
RvStatus RvSipSubsRespondUnauthenticated(
    IN RvSipSubsHandle hSubs,
    IN RvSipNotifyHandle hNotify,
    IN RvUint16 responseCode,
    IN RvChar* strReasonPhrase,
    IN RvSipHeaderType headerType,
    IN void* hHeader);

PARAMETERS

hSubs
The handle to the subscription.

hNotify
The handle to the notification.

responseCode
The response code, either 401 or 407.

strReasonPhrase
The reason phrase. Can be NULL for the default reason phrase.

headerType
The type of the given header
Subscription Authentication API
RvSipSubsRespondUnauthenticated()

**hHeader**
A pointer to the header to be set in the message.

**RETURN VALUES**
Returns RvStatus.
Subscription Authentication API
RvSipSubsRespondUnauthenticatedDigest()

RvSipSubsRespondUnauthenticatedDigest()

DESCRIPTION
Sends a 401/407 response on a SUBSCRIBE, REFER or NOTIFY request. This function builds an Authentication header containing all given parameters, and adds it to the response message.

SYNTAX
RvStatus RvSipSubsRespondUnauthenticatedDigest(
    IN RvSipSubsHandle hSubs,
    IN RvSipNotifyHandle hNotify,
    IN RvUint16 responseCode,
    IN RvChar *strReasonPhrase,
    IN RvChar *strRealm,
    IN RvChar *strDomain,
    IN RvChar *strNonce,
    IN RvChar *strOpaque,
    IN RvBool bStale,
    IN RvSipAuthAlgorithm eAlgorithm,
    IN RvChar *strAlgorithm,
    IN RvSipAuthQopOption eQop,
    IN RvChar *strQop);

PARAMETERS

hSubs
The handle to the subscription.

hNotify
The handle to the notification.

responseCode
The response code, either 401 or 407.
strReasonPhrase
The reason phrase. Can be NULL for the default reason phrase.

strRealm
The realm is mandatory.

strDomain
An optional string that may be NULL.

strNonce
An optional string that may be NULL.

strOpaque
An optional string that may be NULL.

bStale
TRUE or FALSE.

eAlgorithm
The enumeration of the algorithm. If this parameter is set to RVSIP_AUTH_ALGORITHM_OTHER, the algorithm value is taken from the next argument.

strAlgorithm
The string of the algorithm. This parameter will be set only if the eAlgorithm parameter is set to be RVSIP_AUTH_ALGORITHM_OTHER.

eQop
The enumeration of Quality-of-Protection (Qop). If this parameter is set to RVSIP_AUTH_QOP_OTHER, the Qop value will be taken from the next argument.

strQop
The string of Quality-of-Protection (Qop). This parameter will be set only if the eQop parameter is set to RVSIP_AUTH_QOP_OTHER.
Subscription Authentication API

**RETURN VALUES**

Returns RvStatus.
The Subscription DNS functions are:

- RvSipSubsDNSGiveUp()
- RvSipSubsDNSContinue()
- RvSipSubsDNSReSendRequest()
- RvSipSubsDNSGetList()
RvSipSubsDNSGiveUp()

DESCRIPTION
This function is for use in the MSG_SEND_FAILURE state of a subscription or notification. Calling this function stops the sending of the Request message and returns the state machine to its previous state. You can use this function for SUBSCRIBE, REFER or NOTIFY requests. For SUBSCRIBE and REFER messages, use this function if the state was changed to Msg-Send-Failure. In this case, set hNotify to NULL. Use this function for a NOTIFY request if you were informed of the MSG_SEND_FAILURE status in the RvSipSubsNotifyEv() event. In this case, you should supply the notify handle.

SYNTAX
RvStatus RvSipSubsDNSGiveUp(
    IN RvSipSubsHandle hSubs,
    IN RvSipNotifyHandle hNotify);

PARAMETERS
hSubs
The handle to the subscription that sent the request.

hNotify
The handle to the notify object, in the case of a NOTIFY request.

RETURN VALUES
Returns RvStatus.
RvSipSubsDNSContinue()

DESCRIPTION

This function is for use in the MSG_SEND_FAILURE state of a subscription or notification. It creates a new cloned transaction with a new DNS list, and terminates the old transaction. The application may get an outbound message in this stage, and set all the information in it. (This is the same information that was set in the message of the original transaction.) When the message is ready, the application should send the new message to the next IP address with RvSipSubsDNSResendRequest().

You can use this function for SUBSCRIBE, REFER and NOTIFY requests. For SUBSCRIBE and REFER messages, use this function if the state was changed to Msg-Send-Failure. (In this case set hNotify to NULL.) Use this function for a NOTIFY request if you were informed of the MSG_SEND_FAILURE status in the RvSipSubsNotifyEv() event. In this case, you should supply the notify handle.

SYNTAX

RvStatus RvSipSubsDNSContinue(
    IN RvSipSubsHandle hSubs,
    IN RvSipNotifyHandle hNotify);

PARAMETERS

hSubs

The handle to the subscription that sent the request.

hNotify

The handle to the notify object, in the case of a NOTIFY request.

RETURN VALUES

Returns RvStatus.
RvSipSubsDNSReSendRequest()

DESCRIPTION

This function is for use in the MSG_SEND_FAILURE state of a subscription or notification. Calling this function re-sends the cloned transaction to the next IP address and changes back the state of the state machine to the sending state.

Note Before calling this function, you should set all the information that you set in the message in the first sending (such as NOTIFY message body). You can call the RvSipSubsGetOutboundMsg() function and then set your parameters to this message.

You can use this function for SUBSCRIBE, REFER and NOTIFY requests. For SUBSCRIBE and REFER messages, use this function if the state was changed to Msg-Send-Failure. (In this case set hNotify to NULL.) Use this function for a NOTIFY request if you were informed of the MSG_SEND_FAILURE status in the RvSipSubsNotifyEv() event. In this case, you should supply the notify handle.

SYNTAX

RvStatus RvSipSubsDNSReSendRequest(
    IN RvSipSubsHandle hSubs,
    IN RvSipNotifyHandle hNotify);

PARAMETERS

hSubs

The handle to the subscription that sent the request.

hNotify

The handle to the notify object in the case of a NOTIFY request.

RETURN VALUES

Returns RvStatus.
RvSipSubsDNSGetList()

DESCRIPTION

Returns DNS list object from the transaction. You can use this function for SUBSCRIBE, REFER and NOTIFY requests. For SUBSCRIBE and REFER messages, use this function if the state was changed to MSG_SEND_FAILURE. (In this case set hNotify to NULL.) Use this function for a NOTIFY request if you were informed of the MSG_SEND_FAILURE status in the RvSipSubsNotifyEv() event. In this case, you should supply the notify handle.

SYNTAX

RvStatus RvSipSubsDNSGetList(
    IN RvSipSubsHandle hSubs,
    IN RvSipNotifyHandle hNotify,
    OUT RvSipTransportDNSListHandle *phDnsList);

PARAMETERS

hSubs

The handle to the subscription that sent the request.

hNotify

The handle to the notify object in then case of a NOTIFY request.

phDnsList

DNS list handle

RETURN VALUES

Returns RvStatus.
The following group of functions are for REFER implementation. The Subscription REFER API functions are:

- RvSipSubsReferInit()
- RvSipSubsReferInitStr()
- RvSipSubsRefer()
- RvSipSubsReferAccept()
- RvSipNotifySetReferNotifyBody()
- RvSipSubsGetReferToHeader()
- RvSipSubsGetReferredByHeader()
- RvSipSubsGetReferFinalStatus()
RvSipSubsReferInit()

DESCRIPTION
Initiates a REFER subscription with its refer parameters—Refer-To header (mandatory) and Referred-By header. If the subscription was not created within a call-leg, you must first initiate its dialog parameters by calling RvSipSubsDialogInit() before calling this function. This function initializes the REFER subscription, but does not send a REFER request. Use RvSipSubsRefer() to send a REFER request.

SYNTAX
RvStatus RvSipSubsReferInit(
    IN RvSipSubsHandle hSubs,
    IN RvSipReferToHeaderHandle hReferTo,
    IN RvSipReferredByHeaderHandle hReferredBy);

PARAMETERS
hSubs
The handle to the subscription the user wishes to initialize.

hReferTo
The handle to the Refer-To header to set in the subscription.

hReferredBy
The handle to the Referred-By header to set in the subscription. If NULL, a Referred-By header will not be set to the message.

RETURN VALUES
Returns RvStatus.
RvSipSubsReferInitStr()

DESCRIPTION

Initiates a REFER subscription with its REFER parameters in string format—Refer-To header (mandatory) and Referred-By header, and optional Replaces header to be set in the Refer-To header. If the subscription was not created within a call-leg, you must first initiate its dialog parameters by calling RvSipSubsDialogInit() before calling this function. This function initializes the REFER subscription, but does not send a REFER request. Use RvSipSubsRefer() to send a REFER request.

SYNTAX

RvStatus RvSipSubsReferInitStr(
    IN RvSipSubsHandle hSubs,
    IN RvChar* strReferTo,
    IN RvChar* strReferredBy,
    IN RvChar* strReplaces);

PARAMETERS

hSubs

The handle to the subscription the user wishes to initialize.

strReferTo

The string of the Refer-To header value. For example, “A<sip:176.56.23.4:4444;method=INVITE>”

strReferredBy

The string of the Referred-By header value. For example, “<sip:176.56.23.4:4444>“. If NULL, a Referred-By header will not be set to the message.

strReplaces

The Replaces header to be set in the Refer-To header of the REFER request. The Replaces header string must not contain the “Replaces:” prefix. The Replaces header will be kept in the Refer-To header of the subscription.
Subscription Refer API
RvSipSubsReferInitStr()

**RETURN VALUES**

Returns **RvStatus**.
RvSipSubsRefer()

DESCRIPTION
Establishes a REFER subscription by sending a REFER request. This function may be called only after the subscription was initialized with the To, From, Refer-To and optional Referred-By parameters. (If this is a subscription of an already established call-leg, the To and From parameters are taken from the call-leg). Calling this function causes a REFER request to be sent out, and the Subscription state machine to progress to the SUBS_SENT state.

SYNTAX
RvStatus RvSipSubsRefer(
    IN RvSipSubsHandle hSubs);

PARAMETERS
hSubs
The handle to the subscription that the user wishes to refer.

RETURN VALUES
Returns RvStatus.
RvSipSubsReferAccept()

**DESCRIPTION**

Sends a 202 response on a REFER request and updates the Subscription state machine.

Accepting the REFER request creates a new object associated with the REFER subscription. The type of the new object is determined according to the method parameter in the URL of the Refer-To header. If method=SUBSCRIBE or method=REFER, a new subscription is created. If method=INVITE or no method parameter exists in the Refer-To URL, a new call-leg is created. For all other methods, a new transaction is created. The newly created call-leg contains the following parameters:

- Call-Id—the Call-Id of the REFER request
- To—the Refer-To header of the REFER request
- From—the local contact of the REFER subscription.
- Referred-By—the Referred-By header of the REFER request
- Event—in the case of method=SUBSCRIBE
- Refer-To—in the case of method=REFER

This function associates the REFER subscription with only dialog-creating objects (call-leg or subscription). This means that the RvSipSubsReferNotifyReadyEv() callback will be called only if the new object is a call-leg or subscription. In case of a new transaction, the application has to save the association by itself. The application should check the method parameter to know which application object to allocate (and to give in the hAppNewObj parameter), and which handle type to supply to this function as an output parameter. To complete the acceptance of the REFER request, the application must use the newly created object and establish it by calling to RvSipCallLegConnect() in the case of a new call-leg, RvSipSubsSubscribe() in case of a new subscription, or RvSipSubsRefer() in the case of a new REFER subscription.

Note that this function does not send a NOTIFY request. It is the responsibility of the application to create and send an initializing NOTIFY(active) request. (The application can do it from the RvSipSubsReferNotifyReadyEv() callback).
Subscription Refer API
RvSipSubsReferAccept()

**SYNTAX**

```c
RvStatus RvSipSubsReferAccept(
    IN RvSipSubsHandle hSubs,
    IN void* hAppNewObj,
    IN RvSipCommonStackObjectType eForceObjType,
    OUT RvSipCommonStackObjectType *peCreatedObjType,
    OUT void* *phNewObj);
```

**PARAMETERS**

**hSubs**

The handle to the *subscription* the user wishes to accept.

**hAppNewObj**

The handle to the new application object. (*RvSipAppCallLegHandle* in the case of method=INVITE or no method parameter, *RvSipAppSubsHandle* in the case of method=SUBSCRIBE/REFER, or *RvSipAppTranscHandle* in all other cases).

**eForceObjType**

In case the application wants to force the SIP Stack to create a specific object type (for example, create a *transaction* and not a *call-leg* for method=INVITE in the Refer-To URL) it can set the type in this argument. Otherwise it should be *RVSIP_COMMON_STACK_OBJECT_TYPE_UNDEFINED*.

**peCreatedObjType**

The type of object that was created by the SIP Stack. If application gave the *eForceObjType* parameter, it will be equal to *eForceObjType*. Otherwise the value will be according to the method parameter in the Refer-To URL.

**phNewObj**

The handle to the new object that was created by the SIP Stack. (*RvSipCallLegHandle* in the case of method=INVITE or no method parameter, *RvSipSubsHandle* in the case of method=SUBSCRIBE/REFER, or *RvSipTranscHandle* in all other cases).
Subscription Refer API
RvSipSubsReferAccept()

RETURN VALUES

Returns RvStatus.
**RvSipNotifySetReferNotifyBody()**

**DESCRIPTION**
Builds the body for a NOTIFY request of a REFER subscription. This function sets the correct body and Content-Type header in the NOTIFY outbound message.

**SYNTAX**
```
RvStatus RvSipNotifySetReferNotifyBody(
    IN RvSipNotifyHandle hNotify, 
    IN RvInt16 statusCode);
```

**PARAMETERS**
- **hNotify**
  The handle to the notification.
- **statusCode**
  The status code to be set in the NOTIFY message body.

**RETURN VALUES**
Returns RvStatus.
RvSipSubsGetReferToHeader()

DESCRIPTION
Returns the Refer-To header associated with the subscription.

SYNTAX
RvStatus RvSipSubsGetReferToHeader(
    IN RvSipSubsHandle hSubs,
    OUT RvSipReferToHeaderHandle *phReferTo);

PARAMETERS
hSubs
The handle to the subscription.

phReferTo
A pointer to the Refer-To header handle to the subscription.

RETURN VALUES
Returns RvStatus.
Subscription Refer API
RvSipSubsGetReferredByHeader()

---

**RvSipSubsGetReferredByHeader()**

**DESCRIPTION**

Returns the Referred-By header associated with the subscription.

**SYNTAX**

```c
RvStatus RvSipSubsGetReferredByHeader(
    IN   RvSipSubsHandle   hSubs,
    OUT  RvSipReferredByHeaderHandle  *phReferredBy);
```

**PARAMETERS**

- **hSubs**
  
  The handle to the subscription.

- **phReferredBy**
  
  A pointer to the Referred-By header handle to the subscription.

**RETURN VALUES**

Returns `RvStatus`. 
RvSipSubsGetReferFinalStatus()

DESCRIPTION
Returns the status code that was received in the body of the NOTIFY request of a REFER subscription.

SYNTAX
RvStatus RvSipSubsGetReferFinalStatus(
    IN RvSipSubsHandle hSubs,
    OUT RvInt32 *pReferNotifyStatus);

PARAMETERS
hSubs
The handle to the subscription.

pReferNotifyStatus
A pointer to the REFER NOTIFY status value.

RETURN VALUES
Returns RvStatus.
Subscription High Availability API

The Subscription High Availability API functions are:

- RvSipSubsGetActiveSubsStorageSize()
- RvSipSubsStoreActiveSubs()
- RvSipSubsRestoreActiveSubs()
RvSipSubsGetActiveSubsStorageSize()

**DESCRIPTION**

Gets the size of buffer needed to store all parameters of an active *subscription*. (The size of the buffer that should be supplied in RvSipSubsStoreActiveSubs()).

**SYNTAX**

RvStatus RvSipSubsGetActiveSubsStorageSize(
    IN RvSipSubsHandle hSubs,
    OUT RvInt32 *pLen);

**PARAMETERS**

**hSubs**

The handle to the active *subscription* that will be stored.

**pLen**

The size of the buffer needed to store all *subscription* parameters. UNDEFINED in case of failure.

**RETURN VALUES**

Returns RvStatus.
**RvSipSubsStoreActiveSubs()**

**DESCRIPTION**

Copies all subscription parameters from a given subscription to a given buffer. This buffer should be supplied when restoring the subscription. To store subscription information, the subscription must be in the ACTIVE state.

**SYNTAX**

```c
RvStatus RvSipSubsStoreActiveSubs(
    IN RvSipSubsHandle hSubs,
    IN void* memBuff,
    IN RvUint32 buffLen);
```

**PARAMETERS**

- **hSubs**
  
  The handle to the active subscription that will be stored.

- **memBuff**
  
  The buffer that will be filled with the subscription information.

- **buffLen**
  
  The length of the given buffer.

**RETURN VALUES**

Returns `RvStatus`.
RvSipSubsRestoreActiveSubs()

DESCRIPTION

Restore all subscription information into a given subscription. The subscription will assume the ACTIVE state and all subscription parameters will be initialized from the given buffer.

SYNTAX

RvStatus RvSipSubsRestoreActiveSubs(
    IN RvSipSubsHandle hSubs,
    IN void *memBuff,
    IN RvUint32 buffLen);

PARAMETERS

hSubs
The handle to the restored active subscription.

memBuff
The buffer that stores all the subscription information.

buffLen
The buffer size.

RETURN VALUES

Returns RvStatus.
The Subscription Forking API functions are:

- RVsSipsSubsSetForkingEnabledFlag()
- RVsSipsSubsGetForkingEnabledFlag()
- RVsSipsSubsGetOriginalSubs()
**RvSipSubsSetForkingEnabledFlag()**

**DESCRIPTION**

Sets the “forking enabled” flag to the subscription. In case the SUBSCRIBE request was forked by proxy to several notifiers, the subscription may get NOTIFY requests from several notifiers, which are referred to as “Forked NOTIFY requests”. Setting the “forking enabled” flag of the subscription to RV_TRUE will create a new subscription for each Forked NOTIFY request received for the original subscription (the subscription that sent the SUBSCRIBE request).

Setting the “forking enabled” flag of a subscription to RV_FALSE will cause the original subscription to reject with a 481 response (No Matching Call) for every Forked NOTIFY request that is received. For forking details, see sections 3.3.3 and 4.4.9 of RFC 3265.

**SYNTAX**

```c
RvStatus RvSipSubsSetForkingEnabledFlag( 
    IN RvSipSubsHandle hSubs, 
    IN RvBool bForkingEnabled);
```

**PARAMETERS**

- **hSubs**
  
The handle to the subscription that sent the request.

- **bForkingEnabled**
  
The “forking enabled” flag value.

**RETURN VALUES**

Returns RvStatus.
**RvSipSubsGetForkingEnabledFlag()**

**DESCRIPTION**

Gets the “forking enabled” flag from the *subscription*. In case the SUBSCRIBE request was forked by proxy to several notifiers, the *subscription* may get NOTIFY requests from several notifiers, which are referred to as “Forked NOTIFY requests”. Setting the “forking enabled” flag of the *subscription* to RV_TRUE will create a new *subscription* for each Forked NOTIFY request received for the original *subscription* (the *subscription* that sent the SUBSCRIBE request).

Setting the “forking enabled” flag of a *subscription* to RV_FALSE will cause the original *subscription* to reject with a 481 response (No Matching Call) for every Forked NOTIFY request that is received. For forking details, see sections 3.3.3 and 4.4.9 of RFC 3265.

**SYNTAX**

```
RvStatus RvSipSubsGetForkingEnabledFlag(
    IN  RvSipSubsHandle   hSubs,
    OUT RvBool            *pbForkingEnabled);
```

**PARAMETERS**

- **hSubs**
  
The handle to the *subscription* that sent the request.

- **pbForkingEnabled**
  
  A requested flag value.

**RETURN VALUES**

Returns RvStatus.
RvSipSubsGetOriginalSubs() API

**DESCRIPTION**

Gets the handle to the original *subscription*. The original *subscription* is the *subscription* that actually sent the SUBSCRIBE request. In case the SUBSCRIBE request was forked by proxy to several notifiers, the *subscription* may get NOTIFY requests from several notifiers.

If no original *subscription* exists, NULL will be retrieved. This function is intended for use with forked *subscriptions*. For forking details, see sections 3.3.3 and 4.4.9 of RFC 3265.

**SYNTAX**

```c
RvStatus RvSipSubsGetOriginalSubs(
    IN RvSipSubsHandle hSubs,
    OUT RvSipSubsHandle *phOriginalSubs);
```

**PARAMETERS**

- **hSubs**
  The handle to the *subscription* that sent the request.

- **phOriginalSubs**
  A valid pointer which will be updated with the original *subscription* handle.

**RETURN VALUES**

Returns *RvStatus*.
Subscription Forking API
RvSipSubsGetOriginalSubs()
TRANSPORT MODULE

The Transport module allows you to control the name resolution process. This part includes the following section:

- Transport Functions
TRANSPORT FUNCTIONS

The Transport layer of the C SIP Stack allows you to control the sending and receiving of messages over the network. This section contains Transport API functions found in the RvSipTransport.h and RvSipTransportDNS.h header files. The functions included in this section are:

- Transport Functions
- DNS Transport Functions
Transport Functions

TRANSPORT FUNCTIONS

The Transport functions are:

- RvSipTransportMgrSetEvHandlers()
- RvSipTransportMgrGetAppMgrHandle()
- RvSipTransportMgrSetAppMgrHandle()
- RvSipTransportMgrGetStackInstance()
- RvSipTransportMgrCreateConnection()
- RvSipTransportMgrLocalAddressAdd()
- RvSipTransportMgrLocalAddressRemove()
- RvSipTransportMgrLocalAddressFind()
- RvSipTransportMgrLocalAddressGetDetails()
- RvSipTransportMgrLocalAddressGetFirst()
- RvSipTransportMgrLocalAddressGetNext()
- RvSipTransportMgrLocalAddressGetIpTosSockOption()
- RvSipTransportMgrLocalAddressGetIpTosSockOption()
- RvSipTransportConvertStringToIp()
- RvSipTransportConvertIpToString()
- RvSipTransportInjectMsg()
- RvSipTransportConnectionInit()
- RvSipTransportConnectionConnect()
- RvSipTransportConnectionTerminate()
- RvSipTransportConnectionAttachOwner()
- RvSipTransportConnectionDetachOwner()
- RvSipTransportConnectionEnable()
- RvSipTransportConnectionDisable()
- RvSipTransportConnectionIsEnabled()
- RvSipTransportConnectionGetCurrentState()
- RvSipTransportConnectionGetCurrentTlsState()
- RvSipTransportConnectionGetNumOfOwners()
- RvSipTransportConnectionGetTransportType()
- RvSipTransportConnectionGetLocalAddress()
- RvSipTransportConnectionGetRemoteAddress()
- RvSipTransportConnectionGetStackInstance()
- RvSipTransportConnectionTlsHandshake()
Transport Functions

- `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()`
Transport Functions
RvSipTransportMgrSetEvHandlers()

RvSipTransportMgrSetEvHandlers()

DESCRIPTION
Sets the event handlers for all transport events.

SYNTAX
RvStatus RvSipTransportMgrSetEvHandlers(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipAppTransportMgrHandle hAppTransportMgr,
    IN RvSipTransportMgrEvHandlers *pHandlers,
    IN RvUint32 evHandlerStructSize);

PARAMETERS
hTransportMgr
The handle to the TransportMgr.

hAppTransportMgr
An application handle. This handle will be supplied with some transport callbacks.

pHandlers
A pointer to the structure containing application event handler pointers.

evHandlerStructSize
The size of the event handler structure.

RETURN VALUES
Returns RvStatus.
RvSipTransportMgrGetAppMgrHandle()

**DESCRIPTION**

Returns the handle to the application *TransportMgr*. You set this handle in the Stack using the RvSipTransportMgrSetEvHandlers() function.

**SYNTAX**

```c
RvStatus RvSipTransportMgrGetAppMgrHandle(
    IN RvSipTransportMgrHandle hTransportMgr,
    OUT RvSipAppTransportMgrHandle* hAppTransportMgr);
```

**PARAMETERS**

*hTransportMgr*

The handle to the *TransportMgr*.

*hAppTransportMgr*

The application *TransportMgr* handle.

**RETURN VALUES**

Returns RvStatus.
**RvSipTransportMgrSetAppMgrHandle()**

**DESCRIPTION**
Sets the handle to the application `TransportMgr`. You can also supply this handle when calling `RvSipTransportMgrSetEvHandlers()`.

**SYNTAX**

```c
RvStatus RvSipTransportMgrSetAppMgrHandle(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipAppTransportMgrHandle phAppTransportMgr);
```

**PARAMETERS**

- **hTransportMgr**
  The handle to the SIP Stack `TransportMgr`.

- **phAppTransportMgr**
  The application `TransportMgr` handle.

**RETURN VALUES**

Returns `RvStatus`. 
RvSipTransportMgrGetStackInstance()

DESCRIPTION
Returns the handle to the Stack instance to which this TransportMgr belongs.

SYNTAX
RvStatus RvSipTransportMgrGetStackInstance(
    IN RvSipTransportMgrHandle hTransportMgr,
    OUT void* *phStackInstance);

PARAMETERS

hTransportMgr
The handle to the TransportMgr.

phStackInstance
A valid pointer which will be updated with a handle to the Stack instance. To use
this pointer as a Stack handle, you need to cast it to RvSipStackHandle.

RETURN VALUES
Returns RvStatus.
Transport Functions
RvSipTransportMgrCreateConnection()

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.

sizeofEvHanders
The size of the event handler structure.
Transport Functions

RvSipTransportMgrCreateConnection()

**phConn**

The handle to the newly created *connection*.

**RETURN VALUES**

Returns RvStatus.
RvSipTransportMgrLocalAddressAdd()

DESCRIPTION
Adds new local address on which the SIP Stack will receive and send messages. This function does not check if the address to be added is already opened in the SIP Stack.

SYNTAX
RvStatus RvSipTransportMgrLocalAddressAdd(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipTransportAddr *pAddressDetails,
    IN RvUint32 addrStructSize,
    IN RvSipListLocation eLocationInList,
    IN RvSipTransportLocalAddrHandle hBaseLocalAddr,
    OUT RvSipTransportLocalAddrHandle *phLocalAddr);

PARAMETERS

hTransportMgr
The handle to the TransportMgr.

pAddressDetails
A pointer to the memory where the details of the address to be added are stored.

addrStructSize
The size of the structure with details.

eLocationInList
An indication where the new address should be placed in the list of local addresses.

hBaseLocalAddr
An existing address in the list, before or after which the new addresses can be added. The parameter is meaningless, if eLocationInList is not set to RVSIP_PREV_ELEMENT or RVSIP_NEXT_ELEMENT.
Transport Functions
RvSipTransportMgrLocalAddressAdd()

**phLocalAddr**
A pointer to the memory where the handle to the added address will be stored by the function.

**RETURN VALUES**
Returns **RvStatus**.
RvSipTransportMgrLocalAddressRemove()

DESCRIPTION
Removes the local address on which the SIP Stack receives and sends messages. The socket will be closed immediately. Reliable connections that were created to or from the address will be not closed.

SYNTAX
RvStatus RvSipTransportMgrLocalAddressRemove(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipTransportLocalAddrHandle hLocalAddr);

PARAMETERS

hTransportMgr
The handle to the TransportMgr.

hLocalAddr
The handle to the address to be removed.

RETURN VALUES
Returns RvStatus.
RvSipTransportMgrLocalAddressFind()

DESCRIPTION
Finds the local address in the SIP Stack that match the details supplied by pAddressDetails.

SYNTAX
RvStatus RvSipTransportMgrLocalAddressFind(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipTransportAddr *pAddressDetails,
    IN RvUint32 addrStructSize,
    OUT RvSipTransportLocalAddrHandle *phLocalAddr);

PARAMETERS

hTransportMgr
The handle to the TransportMgr.

pAddressDetails
A pointer to the memory where the details of the address to be added are stored.

addrStructSize
The size of the structure with details.

phLocalAddr
A pointer to the memory where the handle to the found address will be stored by the function. NULL will be stored if no matching address was found.

RETURN VALUES
Returns RvStatus.
RvSipTransportMgrLocalAddressGetDetails()

DESCRIPTION
Returns the details of the local address, the handle to which is supplied to the function as a parameter.

SYNTAX
RvStatus RvSipTransportMgrLocalAddressGetDetails(
    IN RvSipTransportLocalAddrHandle hLocalAddr,
    IN RvUint32 addrStructSize,
    OUT RvSipTransportAddr *pAddressDetails);

PARAMETERS
hLocalAddr
The handle to the local address.

addrStructSize
The size of the structure with details.

pAddressDetails
A pointer to the memory where the details of the address to be added are stored.

RETURN VALUES
Returns RvStatus.
RvSipTransportMgrLocalAddressGetFirst()

DESCRIPTION

Gets the handle to the local address. The handle is located at the head of the list of local addresses of the requested transport protocol type.

SYNTAX

RvStatus RvSipTransportMgrLocalAddressGetFirst(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipTransport eTransportType,
    OUT RvSipTransportLocalAddrHandle *phLocalAddr);

PARAMETERS

hTransportMgr
The handle to the TransportMgr.

eTransportType
The type of transport protocol.

phLocalAddr
A pointer to the memory where the handle to the found address will be stored by the function. NULL will be stored if no matching address was found.

RETURN VALUES

Returns RvStatus.
RvSipTransportMgrLocalAddressGetNext()

DESCRIPTION
Gets the handle to the local address, which is located in the list of local addresses next to the base address. The handle to the base address is supplied to the function as a parameter.

SYNTAX
RvStatus RvSipTransportMgrLocalAddressGetNext(
    IN RvSipTransportLocalAddrHandle hBaseLocalAddr,
    OUT RvSipTransportLocalAddrHandle *phLocalAddr);

PARAMETERS
hBaseLocalAddr
The handle to the local address that is located before the requested address.

phLocalAddr
A pointer to the memory, where the handle to the found address will be stored by the function. NULL will be stored, if no matching address was found.

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 *pTypeOfService);
```

**PARAMETERS**

- **hLocalAddr**
  The handle to the local address to be updated.

- **pTypeOfService**
  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**

```c
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.
Transport Functions
RvSipTransportInjectMsg()

**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 state.

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 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 or TLS_CONNECTED, the connection will start a normal disconnection process. TCP connections will move to the CLOSING state. TLS connections will move to the TLS_CLOSE_SEQENSE_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.

pEvHandlers
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.
Transport Functions

RvSipTransportConnectionEnable()

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.
Transport Functions
RvSipTransportConnectionGetCurrentTlsState()

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.
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 or TLS).

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.

strAddress
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.
RvSipTransportConnectionGetStackInstance()

DESCRIPTION
Returns the handle to the SIP Stack instance to which this connection belongs.

SYNTAX
RvStatus RvSipTransportConnectionGetStackInstance(
    IN RvSipTransportConnectionHandle hConn,
    OUT void **phStackInstance);

PARAMETERS
hConn
The handle to the connection.

phStackInstance
A valid pointer which will be updated with a handle to the Stack instance.

RETURN VALUES
Returns RvStatus.
Transport Functions
RvSipTransportConnectionTlsHandshake()

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.
Transport Functions
RvSIPTransportConnectionTlsHandshake()

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

**RETURN VALUES**

Returns RvStatus.
Transport Functions

RvSipTransportConnectionTlsRenegotiate()

**RvSipTransportConnectionTlsRenegotiate()**

**DESCRIPTION**
Starts TLS renegotiation on a connection.

**SYNTAX**

```c
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.
Transport Functions
RvSipTransportConnectionSetAppHandle()

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**

```c
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.
Transport Functions
RvSipTransportTlsGetCertVerificationError()

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.
Transport Functions
RvSipTransportConnectionSetIpTosSockOption()

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.
Transport Functions
RvSipTransportGetIPv4LocalAddressByIndex()

**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**

```c
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 or TLS 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**

```c
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`. 
Transport Functions
RvSipTransportMgrLocalAddressSetAppHandle()

**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.
The Transport DNS functions are:

- RvSipTransportDNSListPushSrvElement()
- RvSipTransportDNSListRemoveTopmostSrvElement()
- RvSipTransportDNSListGetSrvElement()
- RvSipTransportDNSListPopSrvElement()
- RvSipTransportDNSListPushHostElement()
- RvSipTransportDNSListRemoveTopmostHostElement()
- RvSipTransportDNSListGetHostElement()
- RvSipTransportDNSListPopHostElement()
- RvSipTransportDNSListPushIPElement()
- RvSipTransportDNSListRemoveTopmostIPElement()
- RvSipTransportDNSListGetIPElement()
- RvSipTransportDNSListPopIPElement()
- RvSipTransportGetNumberOfDNSListEntries()
- RvSipTransportDNSListGetUsedHostElement()
- RvSipTransportDNSListGetUsedSRVElement()
- RvSipTransportDNSListSetUsedSRVElement()
- RvSipTransportDNSListSetUsedHostElement()
- RvSipTransportDNSListConstruct()
- RvSipTransportDNSListDestruct()
- RvSipTransportDNSListGetSrvElement()
DNS Transport Functions
RvSipTransportDNSListPushSrvElement()

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**

```c
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`. 
DNS Transport Functions
RvSipTransportDNSListPopHostElement()

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.
DNS Transport Functions
RvSipTransportDNSListPushIPElement()

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**

```c
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**

```c
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.
DNS Transport Functions

RvSipTransportDNSListSetUsedHostElement()

**DESCRIPTION**
Sets the host name element that is used to produce the IP list.

**SYNTAX**

```c
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 \textit{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**

\textbf{hTransportMgr}

The handle to the \textit{TransportMgr}.

\textbf{hMemPool}

The handle to the memory pool

\textbf{maxElementsInSingleDnsList}

The maximum number of elements in a single list.

\textbf{phDnsList}

The handle to the DNS list object.

**RETURN VALUES**

Returns \textit{RvStatus}.
RvSipTransportDNSListDestruct()

**DESCRIPTION**
Destructs the Transport DNS list that was built by
RvSipTransportDNSListConstruct().

**SYNTAX**

```c
RvStatus RvSipTransportDNSListDestruct(
    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.
DNS Transport Functions
RvSipTransportDNSListGetSrvElement()

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.
DNS Transport Functions
RvSipTransportDNSListGetSrvElement()
The Resolver module allows you to send DNS queries. This part includes the following section:

- Resolver Functions
RESOLVER FUNCTIONS

The Resolver API functions of the SIP Stack enable you query a DNS server with several DNS Queries—NAPTR, SRV, A and AAAA. The data retrieved from the DNS answer will be stored in a DNS list that the user supplies. This section contains the Resolver API functions found in the RvSipResolver.h header file.

The functions included in this section are:

- Resolver Manager Functions
- Resolver Control Functions
- Resolver Get and Set Functions
Resolver Manager Functions

The Resolver Manager API functions are:

- RvSipResolverMgrSetDnsServers()
- RvSipResolverMgrGetDnsServers()
- RvSipResolverMgrRefreshDnsServers()
- RvSipResolverMgrSetDnsDomains()
- RvSipResolverMgrGetDnsDomains()
- RvSipResolverMgrCreateResolver()
- RvSipResolverMgrClearDnsCache()
- RvSipResolverMgrDumpDnsCache()
Resolver Manager Functions

RvSipResolverMgrSetDnsServers()

DESCRIPTION
Sets a new list of DNS servers to the SIP Stack. In the process of address resolution, host names are resolved to IP addresses by sending DNS queries to the SIP Stack DNS servers. If one server fails, the next DNS server on the list is queried. When the SIP Stack is constructed, a DNS list is set to the SIP Stack using the computer configuration. The application can use this function to provide a new set of DNS servers to the SIP Stack.

Note  This function replaces the current list of servers that the SIP Stack is using.

SYNTAX
RvStatus RvSipResolverMgrSetDnsServers(
    IN RvSipResolverMgrHandle hRslvMgr,
    IN RvSipTransportAddr* pDnsServers,
    IN RvInt32 numOfDnsServers);

PARAMETERS

hRslvMgr
The ResolverMgr.

pDnsServers
A list of addresses of DNS servers to set to the SIP Stack.

numOfDnsServers
The number of DNS servers in the list.

RETURN VALUES
Returns RvStatus.
Resolver Manager Functions
RvSipResolverMgrGetDnsServers()

RvSipResolverMgrGetDnsServers()

DESCRIPTION
Gets the list of the DNS servers that the SIP Stack is using. You can use this function to retrieve the DNS servers with which the SIP Stack was initialized.

SYNTAX
RvStatus RvSipResolverMgrGetDnsServers(
    IN RvSipResolverMgrHandle hRslvMgr,
    IN RvSipTransportAddr* pDnsServers,
    INOUT RvInt32* pNumOfDnsServers);

PARAMETERS
hRslvMgr
The handle to the ResolverMgr.

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

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

RETURN VALUES
Returns RvStatus.
RvSipResolverMgrRefreshDnsServers()

DESCRIPTION

Forces refreshing of the list of DNS servers in the SIP Stack. The application will call this function when a new list of DNS servers has been configured in the system outside of the SIP Stack.

SYNTAX

RvStatus RvSipResolverMgrRefreshDnsServers(
    IN RvSipResolverMgrHandle hRslvMgr)

PARAMETERS

hRslvMgr

The handle to the ResolverMgr.

RETURN VALUES

Returns RvStatus.
**RvSipResolverMgrSetDnsDomains()**

**DESCRIPTION**
Sets a new list of DNS domains to the SIP Stack. The SIP Stack provides 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 to the host name and will query DNS servers. If this is not successful, the system will use the Domain Suffix list to create additional FQDNs in the order listed, and will query DNS servers for each. When the SIP Stack initializes, the DNS domain list is set according to the computer configuration. The application can use this function to provide a new set of DNS domains to the SIP Stack.

**Note**  This function replaces the current Domain list that the SIP Stack is using.

**SYNTAX**

```c
RvStatus RvSipResolverMgrSetDnsDomains(
    IN RvSipResolverMgrHandle hRslvMgr,
    IN RvChar** pDomainList,
    IN RvInt32 numOfDomains);
```

**PARAMETERS**

**hRslvMgr**
The handle to the *ResolverMgr*.

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

**numOfDomains**
The number of domains in the list.
Resolver Manager Functions
RvSipResolverMgrSetDnsDomains()

**RETURN VALUES**

Returns `RvStatus`.
Resolver Manager Functions
RvSipResolverMgrGetDnsDomains()

RvSipResolverMgrGetDnsDomains()

DESCRIPTION
Gets the list of DNS domains from the SIP Stack. This function is useful for determining the list of DNS domains with which the SIP Stack was initialized. To learn more about DNS domains, see RvSipResolverMgrSetDnsDomains().

SYNTAX
RvStatus RvSipResolverMgrGetDnsDomains(
    IN RvSipResolverMgrHandle hRslvMgr,
    INOUT RvChar** pDomainList,
    INOUT RvInt32* pNumOfDomains);

PARAMETERS
hRslvMgr
The handle to the ResolverMgr.

pDomainList
A list of DNS domains (an array of “char” pointers). This array will be filled with pointers to the DNS domains. The list of DNS domains is part of the SIP Stack 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 not be smaller than numOfDomains.

numOfDomains
The size of the pDomainList array. The ResolverMgr will update this parameter with the actual number of domains set in the list.

RETURN VALUES
Returns RvStatus.
**RvSipResolverMgrCreateResolver()**

**DESCRIPTION**

Creates a new *resolver* and exchanges handles with the application. The newly created *resolver* is ready to send DNS queries when `RvSipResolverResolve()` is called.

**SYNTAX**

```c
RvStatus RvSipResolverMgrCreateResolver(
    IN RvSipResolverMgrHandle hRslvMgr,
    IN RvSipAppResolverHandle hAppRslv,
    OUT RvSipResolverHandle* phRslv);
```

**PARAMETERS**

- **hRslvMgr**
  The handle to the *ResolverMgr*.

- **hAppRslv**
  The application handle to the newly created *resolver*.

- **phRslv**
  A pointer to the SIP Stack handle of the *resolver*.

**RETURN VALUES**

Returns `RvStatus`. 
Resolver Manager Functions
RvSipResolverMgrClearDnsCache()

**RvSipResolverMgrClearDnsCache()**

**DESCRIPTION**
Resets the SIP Stack DNS cache.

**SYNTAX**
RvStatus RvSipResolverMgrClearDnsCache(
    IN RvSipResolverMgrHandle hRslvMgr);

**PARAMETERS**

**hRslvMgr**
The handle to the *ResolverMgr*.

**RETURN VALUES**
Returns *RvStatus*. 
**RvSipResolverMgrDumpDnsCache()**

**DESCRIPTION**

Dumps the SIP Stack DNS cache data into a file.

**SYNTAX**

```c
RvStatus RvSipResolverMgrDumpDnsCache(
    IN RvSipResolverMgrHandle hRslvMgr,
    IN RvCha *strDumpFile);
```

**PARAMETERS**

- **hRslvMgr**
  
  The handle to the `ResolverMgr`.

- **strDumpFile**
  
  The name of the file into which to dump the cache data.

**RETURN VALUES**

Returns `RvStatus`.
Resolver Control Functions

The Resolver Control functions are:

- RvSipResolverResolve()
- RvSipResolverTerminate()
Resolver Control Functions

RvSipResolverResolve()

**DESCRIPTION**

Resolves a name according to a scheme. When calling this function, the resolver starts a DNS algorithm. The algorithm will try to obtain information from the configured DNS server using one or more DNS queries. The different algorithms are defined in the eMode parameter that is supplied to this function. For example, if the supplied mode is RVSIP_RESOLVER_MODE_FIND_IP_BY_HOST, the resolver will try to find the IP of a specific host. The resolver will try both IPv4 and IPv6 using A and AAAA queries respectively. The data retrieved from the DNS server is stored in the DNS list supplied to this function as a parameter. Since DNS activities are a-synchronous, the resolver will notify the DNS query results using a callback function. The callback function must also be supplied to this function.

**Note** A single resolver can perform only a single algorithm at any given time.

**SYNTAX**

```c
RvStatus RvSipResolverResolve(
    IN RvSipResolverHandle hRslv,
    IN RvSipResolverMode eMode,
    IN RvChar* strQueryString,
    IN RvSipResolverScheme eScheme,
    IN RvBool bIsSecure,
    IN RvUint16 knownPort,
    IN RvSipTransport knownTransport,
    IN RvSipTransportDNSListHandle hDns,
    IN RvSipResolverReportDataEv pfnResolveCB);
```

**PARAMETERS**

- **hRslv**
  The resolver handle.
Resolver Control Functions
RvSipResolverResolve()

**eMode**
The resolving mode that specifies the DNS algorithm that will be activated.

**strQueryString**
The string to resolve (for example, “host.com”). It is the responsibility of the application to keep the query string until the query has ended.

**eScheme**
The required scheme, such as “im”, “sip”, and so on.

**bIsSecure**
The resolution restricted to TLS only. You can use this parameter to indicate that the any record that does not represent a secure transport will be discarded.

**knownPort**
The port for queries that do not retrieve the port. This port will be placed in the DNS list along with the DNS results. For example, A records do not contain a port.

**knownTransport**
The transport for queries that do not retrieve the transport. This transport will be placed in the DNS list along with the DNS results. For example, SRV records do not contain a transport.

**hDns**
A DNS list. The resolver will place all DNS results in this list along with additional information, such as the knownPort and knownTransport values received as parameters to this function.

**pfnResolveCB**
A function pointer that the resolver will use for reporting results.

**RETURN VALUES**
Returns RvStatus.
RvSipResolverTerminate()

DESCRIPTION
Terminates a resolver. If the resolver has a pending query, the query is canceled and a result will not be reported.

SYNTAX
RvStatus RvSipResolverTerminate(
    IN RvSipResolverHandle hRslv);

PARAMETERS
hRslv
The resolver handle.

RETURN VALUES
Returns RvStatus.
Resolver Get and Set Functions

**RESOLVER GET AND SET FUNCTIONS**

The Resolver Get and Set function is:

- RvSipResolverGetDnsList()
Resolver Get and Set Functions
RvSipResolverGetDnsList()

RvSipResolverGetDnsList()

DESCRIPTION
Gets the DNS list from the resolver. The list contains the results for the DNS queries.

SYNTAX
RvStatus RvSipResolverGetDnsList(
    IN  RvSipResolverHandle hRslv,
    OUT RvSipTransportDNSListHandle *phdns);

PARAMETERS
hRslv
The resolver handle.

phdns
The list handle to the DNS.

RETURN VALUES
Returns RvStatus.
Resolver Get and Set Functions
RvSipResolverGetDnsList()
MEMORY POOL MODULE

This part includes the following section:

- RPOOL Functions
RPOOL FUNCTIONS

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, allocate pages and read and write from memory pool and allocated pages. This section contains RPOOL functions found in the rpool_API.h header file.

The functions included in this section are:

- 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 Control Functions

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.
RPOOL Control Functions
RPOOL_Construct()

**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**

```c
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 Control Functions

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 Control Functions
RPOOL_CopyToExternal()

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 Control Functions
RPOOL_AppendFromExternalToPage()

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 than 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 Control Functions**

**RPOOL_GetResources()**

**DESCRIPTION**

Returns the resources used by a given pool.

**SYNTAX**

```c
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.

- **pMaxUsage**
  
The maximum number of concurrently-used blocks in the pool until the time this function was called.

**RETURN VALUES**

- None.
RPOOL Control Functions

RPOOL_GetResources()
MID-LAYER MODULE

The Mid-layer API functions of the Mid-layer module interact with OS Abstraction layer (Common Core) functionality, including select/poll and timers.

This part includes the following section:

- Mid-layer Functions
The Mid-layer API 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 stacks only, since only these threads use the select() loop.

This section contains Mid-layer API functions found in the RvSipMid.h header file.

The functions included in this section are:

- Mid-layer Control Functions
The Mid-layer Control functions are:

- RvSipMidInit()
- RvSipMidEnd()
- RvSipMidConstruct()
- RvSipMidPrepareDestruct()
- RvSipMidDestruct()
- RvSipMidSetLog()
- RvSipMidMemAlloc()
- RvSipMidMemFree()
- RvSipMidTimeInMilliGet()
- RvSipMidTimeInSecondsGet()
- RvSipMidTimerSet()
- RvSipMidTimerReset()
- RvSipMidSelectCallOn()
- RvSipMidSelectSetMaxDescs()
- RvSipMidSelectGetMaxDesc()
- RvSipMidSelectGetEventsRegistration()
- RvSipMidSelectEventsHandling()
- RvSipMidPollGetEventsRegistration()
- RvSipMidPollEventsHandling()
- RvSipMidEncodeB64()
- RvSipMidDecodeB64()
- RvSipMidTlsSetLockingCallback()
- RvSipMidTlsSetThreadIdCallback()
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.
**RvSipMidEnd()**

**DESCRIPTION**
Ends the Mid-layer services.

**SYNTAX**

```c
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

```c
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*. 
Mid-layer Control Functions
RvSipMidPrepareDestruct()

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 perform 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 was initiated before the SIP Stack. Use `RvSipStackGetLogHandle()` to get the log handle from the SIP Stack.

### SYNTAX

```c
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**  
```c  
void RvSipMidMemFree(  
    IN void* memptr);  
```

**PARAMETERS**  

`memptr`  
The memory to free.  

**RETURN VALUES**  
None.
**RvSipMidTimeInMilliGet()**

**DESCRIPTION**
Gets a timestamp value in milliseconds.

**SYNTAX**
```c
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**

```c
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.
Mid-layer Control Functions
RvSipMidSelectCallOn()

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**

```c
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 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 events for sockets opened by the SIP Stack.

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.

drSet
Not in use from C SIP Stack v.4.5. GA3.

wrSet
Not in use from C SIP Stack v.4.5. GA3.

exSet
Not in use from C SIP Stack v.4.5. GA3.

numFds
Not in use from C SIP Stack v.4.5. GA3.

numEvents
Not in use from C SIP Stack v.4.5. GA3.
**Mid-layer Control Functions**

RvSipMidSelectEventsHandling()

**RETURN VALUES**

Returns RvStatus.

**REMARKS**

This function can be called only in the thread that initiated the SIP Stack.
Mid-layer Control Functions
RvSipMidPollGetEventsRegistration()

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.
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.
RvSipMidEncodeB64()

**DESCRIPTION**
Performs a 64 bit encoding operation of a given number of bytes in a given buffer.

**SYNTAX**
```c
RvUint32 RvSipMidEncodeB64(
    IN RvUint8* inTxt,
    IN RvInt inLen,
    INOUT RvUint8* outTxt,
    IN RvInt outLen);
```

**PARAMETERS**
- **inTxt**
The buffer to be encoded.
- **inLen**
The 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.
RvSipMidTlsSetLockingCallback()

**DESCRIPTION**

The 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 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 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 SIP Stack implementation. Note that the implementation will not be available after the SIP Stack is destructed. To activate or stop the 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.
Mid-layer Control Functions
RvSipMidTlsSetThreadIdCallback()

**DESCRIPTION**

The 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 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 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 SIP Stack implementation. Note that the implementation will not be available after the SIP Stack is destructed. To activate or stop the 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**

```c
RvStatus RvSipMidTlsSetThreadIdCallback(
    RvBool bSet);
```

**PARAMETERS**

- **bSet**
  
  If RV_TRUE, the callback will be set. Otherwise the callback will be removed.

**RETURN VALUES**

Returns RvStatus.
COMMON MODULE

The Common Module contains utility functions for handling lists. This part includes the following section:

- Common List Functions
COMMON LIST FUNCTIONS

The Common List is a general structure which enables you to manage a list. Each element in the list is identified by its type and data. This section contains an API implementation of the SIP Common List found in the RvSipCommonList.h header file.

The functions included in this section are:
- Common List Control Functions
- Common List Get and Set Functions
Common List Control Functions

The Common List Control functions are:

- RvSipCommonListConstruct()
- RvSipCommonListConstructOnPage()
- RvSipCommonListDestruct()
- RvSipCommonListPushElem()
- RvSipCommonListRemoveElem()
RvSipCommonListConstruct()

DESCRIPTION
Constructs a sip-common-list object (sip-common-list).

SYNTAX
RvStatus RvSipCommonListConstruct(
    IN HRPOOL hPool,
    IN RV_LOG_Handle hLog,
    OUT RV_SipCommonListHandle phList
);

PARAMETERS

hPool
The handle to the list pool.

hLog
The handle to the SIP Stack Log Module.

phList
The handle to the created list.

RETURN VALUES
Returns RvStatus.
RvSipCommonListConstructOnPage()

DESCRIPTION
Constructs a *sip-common-list* on a given pool and page.

SYNTAX

```c
RvStatus RvSipCommonListConstructOnPage(
    IN HRPOOL hPool,
    IN HPAGE hPage,
    IN RV_LOG_Handle hLog,
    OUT RvSipCommonListHandle* phList);
```

PARAMETERS

- **hPool**
  The handle to the list pool.

- **hPage**
  The handle to the list page.

- **hLog**
  The handle to the SIP Stack Log Module.

- **phList**
  The handle to the created list.

RETURN VALUES

Returns *RvStatus*. 
RvSipCommonListDestruct()

DESCRIPTION
Destructs a \textit{sip-common-list} and frees all stored data (headers and messages).

SYNTAX
RvStatus RvSipCommonListDestruct(
    IN RvSipCommonListHandle hList);

PARAMETERS
hList
The handle to the list to destruct.

RETURN VALUES
Returns \textit{RvStatus}.
Common List Control Functions
RvSipCommonListPushElem()

**RvSipCommonListPushElem()**

**DESCRIPTION**

Adds a given element to the *sip-common-list*. First, the application must allocate the new element on the *sip-common-list* page, and then call this function to add the allocated element to the *sip-common-list*. In this function, the application gives the new element type and data. The type is an integer value which will be given back in the `RvSipCommonListGetElem()` function. The data is the pointer to the new allocated element on the *sip-common-list* page, which will also be given back in the `RvSipCommonListGetElem()` function.

**SYNTAX**

```c
RvStatus RvSipCommonListPushElem(
    IN RvSipCommonListHandle hList,
    IN RvInt eElemType,
    IN void* pData,
    IN RvSipListLocation eLocation,
    IN RvSipCommonListElemHandle hPos,
    INOUT RvSipCommonListElemHandle* phNewPos);
```

**PARAMETERS**

- **hList**
  The handle to the list.

- **eElemType**
  An integer value that describes the element type. This value will be given back in `RvSipCommonListGetElem()` (For example, `RVSIP_HEADERTYPE_CONTACT` when adding a contact header to the list)

- **pData**
  A pointer to the allocated element data. (For example, this is the Contact header handle that the application got when calling `RvSipContactHeaderConstruct`).

- **eLocation**
  The inserted element location (first, last, and so on).
Common List Control Functions

RvSipCommonListPushElem()

**hPos**
The current list position. This value is relevant for the next or previous location.

**pNewPos**
The returned location of the object that was pushed.

**RETURN VALUES**
Returns RvStatus.
RvSipCommonListRemoveElem()

DESCRIPTION
Removes an element from the sip-common-list.

SYNTAX
RvStatus RvSipCommonListRemoveElem(
    IN RvSipCommonListHandle hList,
    IN RvSipCommonListElemHandle hListElem);

PARAMETERS

hList
The handle to the sip-common-list.

hListElem
The handle to the sip-common-list element to be removed.

RETURN VALUES
Returns RvStatus.
Common List Get and Set Functions
RvSipCommonListRemoveElem()
Common List Get and Set Functions
RvSipCommonListGetElem()

RvSipCommonListGetElem()

DESCRIPTION
Returns an element from the sip-common-list. This function returns the first or last elements, or the previous or following elements relative to another element.

SYNTAX
RvStatus RvSipCommonListGetElem(
    IN RvSipCommonListHandle hList,
    IN RvSipListLocation eLocation,
    IN RvSipCommonListElemHandle hPos,
    OUT RvInt *peElemType,
    OUT void* *pElemData,
    OUT RvSipCommonListElemHandle *phNewPos);

PARAMETERS

hList
The handle to the sip-common-list.

eLocation
The location of the desired element in the sip-common-list (first, last, next or previous).

hPos
The current sip-common-list position. This value is relevant if location is next or previous.

peElemType
The type of the sip-common-list element that was returned from list. (This is the element type that you set in the RvSipCommonListPushElem() function).

pElemData
The data of the sip-common-list element that was returned from list. (This is the data that you set in the RvSipCommonListPushElem() function).
Common List Get and Set Functions
RvSipCommonListGetElem()

**phNewPos**
The location of the object in the list that was returned.

**RETURN VALUES**
Returns *RvStatus*. 
RvSipCommonListIsEmpty()

DESCRIPTION
Indicates whether or not the sip-common-list is empty.

SYNTAX
RvStatus RvSipCommonListIsEmpty(
    IN  RvSipCommonListHandle  hList,
    OUT RvBool*               pbIsEmpty);

PARAMETERS

hList
The handle to the sip-common-list.

pbIsEmpty
RV_TRUE if the sip-common-list is empty. Otherwise, RV_FALSE.

RETURN VALUES
Returns RvStatus.
RvSipCommonListGetPool()

**DESCRIPTION**
Returns the sip-common-list memory pool.

**SYNTAX**
```
HRPOOL RvSipCommonListGetPool(
    IN RvSipCommonListHandle hList);
```

**PARAMETERS**

- **hList**
  The handle to the sip-common-list.

**RETURN VALUES**
Returns the memory pool handle.
RvSipCommonListGetPage()

DESCRIPTION
Return the sip-common-list memory page.

SYNTAX
HPAGE RvSipCommonListGetPage(
    IN RvSipCommonListHandle hList);

PARAMETERS
hList
The handle to the sip-common-list.

RETURN VALUES
Returns the memory page handle.
TYPE DEFINITIONS AND STATUS CODES

This part includes the following sections:

- Common Type Definitions
- Stack API Type Definitions
- Call-leg API Type Definitions
- Transaction API Type Definitions
- Transmitter API Type Definitions
- Register-Client API Type Definitions
- Authenticator API Type Definitions
- Event Notification API Type Definitions
- Transport API Type Definitions
- Resolver API Type Definitions
- RPOOL Type Definitions
- Mid-layer Type Definitions
- Resource Type Definitions
- Status Codes
This section includes the SIP Stack Call-leg type definitions and callback functions defined in the RvSipCallLegTypes.h header file.

This section includes:
- Handle Type Definitions
- Call-leg Type Definitions
- Call-leg Callback Functions
Handle Type Definitions

The handle type definitions are:

- RvSipCallLegMgrHandle
- RvSipCallLegHandle
- RvSipAppCallLegHandle
- RvSipCallLegInviteHandle
- RvSipAppCallLegInviteHandle
**RvSipCallLegMgrHandle**

**DESCRIPTION**

The declaration of a handle to a Call-legMgr. The Call-legMgr manages all call-legs and the handle is needed for all Call-leg Manager API actions, such as the creation of a call-leg.

**SYNTAX**

```c
RV_DECLARE_HANDLE(RvSipCallLegMgrHandle);
```
Handle Type Definitions
RvSipCallLegHandle

RvSipCallLegHandle

DESCRIPTION
The declaration of a call-leg handle. A call-leg handle is required by all Call-leg API functions and is used to reference the call-leg.

SYNTAX

RV_DECLARE_HANDLE(RvSipCallLegHandle);
**RvSipAppCallLegHandle**

**DESCRIPTION**

The declaration of an application handle to a *call-leg*. The application uses this handle to associate SIP Stack *call-legs* with application *call-legs*. The application gives the application handle when a new *call-leg* is created. The SIP Stack will give this handle back to the application in every callback function.

**SYNTAX**

```c
RV_DECLARE_HANDLE(RvSipAppCallLegHandle);
```
Handle Type Definitions
RvSipCallLegInviteHandle

RvSipCallLegInviteHandle

DESCRIPTION
The declaration of an Invite object handle. An Invite object is needed in all call-leg re-Invite functions and is used to reference the correct re-Invite object.

SYNTAX
RV_DECLARE_HANDLE(RvSipCallLegInviteHandle);
RvSipAppCallLegInviteHandle

DESCRIPTION
The declaration of an application handle to an INVITE object. The application uses this handle to associate a SIP Stack INVITE object with an application INVITE object. The application gives the application handle when a new INVITE object is created. The SIP Stack will give this handle back to the application in every callback function.

SYNTAX

RV_DECLARE_HANDLE(RvSipAppCallLegInviteHandle);
CALL-LEG TYPE DEFINITIONS

The Call-leg type definitions are:

- RvSipCallLegState
- RvSipCallLegStateChangeReason
- RvSipCallLegModifyState
- RvSipCallLegDirection
- RvSipCallLegPrackState
- RvSipCallLegTranscState
- RvSipCallLegByeState
- RvSipCallLegReplacesStatus
- RvSipCallLegReplacesReason
- RvSipCallLegSessionTimerNotificationReason
- RvSipCallLegSessionTimerNegotiationReason
- RvSipCallLegSessionTimerRefresherPreference
- RvSipCallLegHARestoreMode
- RvSipCallLegEvHandlers
RvSipCallLegState

DESCRIPTION

RvSipCallLegState represents a state in a session. The state changes are reported using the RvSipCallLegStateChangedEv() callback function together with a state change reason that indicates to the application how the new state was reached.

Note For a detailed description of the call-leg states, see the Working with Call-legs (Dialogs) chapter in the HP-UX C SIP Stack Programmer’s Guide.

SYNTAX

typedef enum{
    RVSIP_CALL_LEG_STATE_UNDEFINED = -1,
    RVSIP_CALL_LEG_STATE_IDLE,
    RVSIP_CALL_LEG_STATE_INVITING,
    RVSIP_CALL_LEG_STATE_REDIRECTED,
    RVSIP_CALL_LEG_STATE_UNAUTHENTICATED,
    RVSIP_CALL_LEG_STATE_OFFERING,
    RVSIP_CALL_LEG_STATE_ACCEPTED,
    RVSIP_CALL_LEG_STATE_CONNECTED,
    RVSIP_CALL_LEG_STATE_DISCONNECTED,
    RVSIP_CALL_LEG_STATE_DISCONNECTING,
    RVSIP_CALL_LEG_STATE_TERMINATED,
    RVSIP_CALL_LEG_STATE_REMOTE_ACCEPTED,
    RVSIP_CALL_LEG_STATE_CANCELLING,
    RVSIP_CALL_LEG_STATE_PROCEEDING,
    RVSIP_CALL_LEG_STATE_PROCEEDING_TIMEOUT,
    RVSIP_CALL_LEG_STATE_MSG_SEND_FAILURE
}RvSipCallLegState;
RvSipCallLegStateChangeReason

DESCRIPTION

A call-leg object has an inner state. The state changes according to the state machine defined for the call-leg. Each state change is explained by one of the reasons enumerated below.

SYNTAX

typedef enum {
    RVSIP_CALL_LEG_REASON_UNDEFINED = -1,
    RVSIP_CALL_LEG_REASON_LOCAL_INVITING,
    RVSIP_CALL_LEG_REASON_REMOTE_INVITING,
    RVSIP_CALL_LEG_REASON_LOCAL_REFER,
    RVSIP_CALL_LEG_REASON_REMOTE_REFER,
    RVSIP_CALL_LEG_REASON_LOCAL_REFER_NOTIFY,
    RVSIP_CALL_LEG_REASON_REMOTE_REFER_NOTIFY,
    RVSIP_CALL_LEG_REASON_LOCAL_ACCEPTED,
    RVSIP_CALL_LEG_REASON_REMOTE_ACCEPTED,
    RVSIP_CALL_LEG_REASON_REMOTE_ACK,
    RVSIP_CALL_LEG_REASON_REDIRECTED,
    RVSIP_CALL_LEG_REASON_LOCAL_REJECT,
    RVSIP_CALL_LEG_REASON_REQUEST_FAILURE,
    RVSIP_CALL_LEG_REASON_SERVER_FAILURE,
    RVSIP_CALL_LEG_REASON_GLOBAL_FAILURE,
    RVSIP_CALL_LEG_REASON_LOCAL_DISCONNECTING,
    RVSIP_CALL_LEG_REASON_DISCONNECTED,
    RVSIP_CALL_LEG_REASON_REMOTE_DISCONNECTED,
    RVSIP_CALL_LEG_REASON_LOCAL_FAILURE,
    RVSIP_CALL_LEG_REASON_LOCAL_TIME_OUT,
    RVSIP_CALL_LEG_REASON_CALL_TERMINATED,
    RVSIP_CALL_LEG_REASON_AUTH_NEEDED,
    RVSIP_CALL_LEG_REASON_UNSUPPORTED_AUTH_PARAMS,
    RVSIP_CALL_LEG_REASON_LOCAL_CANCELING,
    RVSIP_CALL_LEG_REASON_REMOTE_CANCELED,
    RVSIP_CALL_LEG_REASON_ACK_SENT,
}
RvSipCallLegStateChangeReason

RVSIP_CALL_LEG_REASON_CALL_CONNECTED,
RVSIP_CALL_LEG_REASON_REMOTE_PROVISIONAL_RESP,
RVSIP_CALL_LEG_REASON_REMOTE_REFER_REPLACES,
RVSIP_CALL_LEG_REASON_REMOTE_INVITING_REPLACES,
RVSIP_CALL_LEG_REASON_REMOTE_DISCONNECT_REQUESTED,
RVSIP_CALL_LEG_REASON_DISCONNECT_LOCAL_REJECT,
RVSIP_CALL_LEG_REASON_DISCONNECT_REMOTE_REJECT,
RVSIP_CALL_LEG_REASON_DISCONNECT_LOCAL_ACCEPT,
RVSIP_CALL_LEG_REASON_NETWORK_ERROR,
RVSIP_CALL_LEG_REASON_503_RECEIVED,
RVSIP_CALL_LEG_REASON_GIVE_UP_DNS,
RVSIP_CALL_LEG_REASON_CONTINUE_DNS,
RVSIP_CALL_LEG_REASON_OUT_OF_RESOURCES,
RVSIP_CALL_LEG_REASON_FORKED_CALL_NO_FINAL_RESPONSE,
}

PARAMETERS

RVSIP_CALL_LEG_REASON_UNDEFINED
The reason is undefined.

RVSIP_CALL_LEG_REASON_LOCAL_INVITING
The local party invited the remote party.

RVSIP_CALL_LEG_REASON_REMOTE_INVITING
The remote party invited the local party.

RVSIP_CALL_LEG_REASON_LOCAL_REFER
The local party sent a REFER message to the remote party.

RVSIP_CALL_LEG_REASON_REMOTE_REFER
The remote party sent a REFER message to the local party.

RVSIP_CALL_LEG_REASON_LOCAL_REFER_NOTIFY
The local party sent a NOTIFY message to the remote party.
Call-leg Type Definitions
RvSipCallLegStateChangeReason

**RVSIP_CALL_LEG_REASON_REMOTE_REFER_NOTIFY**
The remote party sent a NOTIFY message to the local party.

**RVSIP_CALL_LEG_REASON_LOCAL_ACCEPTED**
The local party accepted the call.

**RVSIP_CALL_LEG_REASON_REMOTE_ACCEPTED**
The remote party accepted the call.

**RVSIP_CALL_LEG_REASON_REMOTE_ACK**
The remote party sent an ACK.

**RVSIP_CALL_LEG_REASON_REDIRECTED**
A 3XX response was received.

**RVSIP_CALL_LEG_REASON_LOCAL_REJECT**
The local party rejected the call.

**RVSIP_CALL_LEG_REASON_REQUEST_FAILURE**
A 4XX response was received—does not include 401 and 407 response.

**RVSIP_CALL_LEG_REASON_SERVER_FAILURE**
A 5XX response was received.

**RVSIP_CALL_LEG_REASON_GLOBAL_FAILURE**
A 6XX response was received.

**RVSIP_CALL_LEG_REASON_LOCAL_DISCONNECTING**
The local party sent a BYE request and is waiting for response.

**RVSIP_CALL_LEG_REASON_DISCONNECTED**
The local party received a final response to a previously sent BYE request.

**RVSIP_CALL_LEG_REASON_REMOTE_DISCONNECTED**
The remote party sent a BYE request.
RVSIP_CALL_LEG_REASON_LOCAL_FAILURE
The local party failed. For example, the local party failed to send a request or encountered a resources problem.

RVSIP_CALL_LEG_REASON_LOCAL_TIME_OUT
A timeout occurred. For example, an expected response did not arrived.

RVSIP_CALL_LEG_REASON_CALL_TERMINATED
The call-leg was terminated.

RVSIP_CALL_LEG_REASON_AUTH_NEEDED
A 401 or 407 response was received.

RVSIP_CALL_LEG_REASON_UNSUPPORTED_AUTH_PARAMS
A 401 or 407 response was received but the authentication parameters are not supported.

RVSIP_CALL_LEG_REASON_LOCAL_CANCELLING
The local party cancelled the outgoing call.

RVSIP_CALL_LEG_REASON_REMOTE_CANCELLED
The remote party cancelled the incoming call.

RVSIP_CALL_LEG_REASON_ACK_SENT
The local party sent an ACK message.

RVSIP_CALL_LEG_REASON_CALL_CONNECTED
The call is connected and a re-INVITE process can start.

RVSIP_CALL_LEG_REASON_REMOTE_PROVISIONAL_RESP
The remote party sent a provisional response to the INVITE or re-INVITE request
Call-leg Type Definitions
RvSipCallLegStateChangeReason

RVSIP_CALL_LEG_REASON_REMOTE_REFER_REPLACES
The remote party sent a REFER request with Replaces header inside the Refer-To header.

RVSIP_CALL_LEG_REASON_REMOTE_INVITING_REPLACES
The remote party sent an INVITE request with Replaces header.

RVSIP_CALL_LEG_REASON_REMOTE_DISCONNECT_REQUESTED
The remote party sent a BYE request.

RVSIP_CALL_LEG_REASON_DISCONNECT_LOCAL_REJECT
The local party rejected the BYE request it received.

RVSIP_CALL_LEG_REASON_DISCONNECT_REMOTE_REJECT
The remote party rejected the BYE request it received.

RVSIP_CALL_LEG_REASON_DISCONNECT_LOCAL_ACCEPT
The local party accepted the BYE request it received.

RVSIP_CALL_LEG_REASON_NETWORK_ERROR
One of the reasons for moving to the MSG_SEND_FAILURE state. The message failed to be sent due to a network error.

RVSIP_CALL_LEG_REASON_503_RECEIVED
One of the reasons for moving to the MSG_SEND_FAILURE state. The message received a 503 response.

RVSIP_CALL_LEG_REASON_GIVE_UP_DNS
The local party decided to give up sending a message after its previous send attempt failed (DNS feature).

RVSIP_CALL_LEG_REASON_CONTINUE_DNS
The local party decided to send a message to the next address in its transaction DNS list, after its previous send attempt failed (DNS feature).
RVSIP_CALL_LEG_REASON_OUT_OF_RESOURCES
An out-of-resource problem.

RVSIP_CALL_LEG_REASON_FORKED_CALL_NO_FINAL_RESPONSE
A call-leg that was created by an hx response on a forked request did not get a final response.
Call-leg Type Definitions
RvSipCallLegModifyState

RvSipCallLegModifyState

DESCRIPTION
Represents a state of a re-INVITE process in a connected session.

Note For a detailed description of the call-leg modify states, see the Working with Call-legs (Dialogs) chapter in the HP-UX C SIP Stack Programmer’s Guide.

SYNTAX

typedef enum(
    RVSIP_CALL_LEG_MODIFY_STATE_UNDEFINED = -1,
    RVSIP_CALL_LEG_MODIFY_STATE_IDLE,
    RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_RCVD,
    RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_RESPONSE_SENT,
    RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_SENT,
    RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_RESPONSE_RCVD,
    RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_REMOTE_ACCEPTED,
    RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_CANCELLING,
    RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_PROCEEDING,
    RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_CANCELLED,
    RVSIP_CALL_LEG_MODIFY_STATE_REINVITE_PROCEEDING_TIMEOUT,
    RVSIP_CALL_LEG_MODIFY_STATE_MSG_SEND_FAILURE,
    RVSIP_CALL_LEG_MODIFY_STATE_ACK_SENT,
    RVSIP_CALL_LEG_MODIFY_STATE_ACK_RCVD,
    RVSIP_CALL_LEG_MODIFY_STATE_TERMINATED
)RvSipCallLegModifyState;
RvSipCallLegDirection

**DESCRIPTION**

Indicates whether the *call-leg* represents an incoming or an outgoing call.

**SYNTAX**

```c
typedef enum{
    RVSIP_CALL_LEG_DIRECTION_UNDEFINED = -1,
    RVSIP_CALL_LEG_DIRECTION_INCOMING,
    RVSIP_CALL_LEG_DIRECTION_OUTGOING
} RvSipCallLegDirection;
```
**RvSipCallLegPrackState**

**DESCRIPTION**
Indicates the status of a PRACK process. A PRACK process can start in two cases:
- When the application calls the `RvSipCallLegProvisionalResponseReliable()` function.
- When a reliable provisional response is received by the call-leg.

The PRACK process is automated by the Transaction layer and involves the PRACK request and response. The `RvSipCallLegPrackStateChangedEv()` callback is only a notification to the application about the PRACK process and no response is expected for the different states.

**SYNTAX**
```
typedef enum{
    RVSIP_CALL_LEG_PRACK_STATE_UNDEFINED = -1,
    RVSIP_CALL_LEG_PRACK_STATE_REL_PROV_RESPONSE_RCVD,
    RVSIP_CALL_LEG_PRACK_STATE_PRACK_SENT,
    RVSIP_CALL_LEG_PRACK_STATE_PRACK_FINAL_RESPONSE_RCVD,
    RVSIP_CALL_LEG_PRACK_STATE_PRACK_RCVD,
    RVSIP_CALL_LEG_PRACK_STATE_PRACK_FINAL_RESPONSE_SENT
}RvSipCallLegPrackState;
```

**PARAMETERS**

**RVSIP_CALL_LEG_PRACK_STATE_UNDEFINED**
No defined state.

**RVSIP_CALL_LEG_PRACK_STATE_REL_PROV_RESPONSE_RCVD**
The call-leg received a reliable provisional response.

**RVSIP_CALL_LEG_PRACK_STATE_PRACK_SENT**
A PRACK request was automatically sent for this call-leg.
RVSIP_CALL_LEG_PRACK_STATE_PRACK_FINAL_RESPONSE_RCVD
A final response was received for a previously sent PRACK.

RVSIP_CALL_LEG_PRACK_STATE_PRACK_PRACK_RCVD
The call-leg received a PRACK after sending a reliable provisional response.

RVSIP_CALL_LEG_PRACK_STATE_PRACK_FINAL_RESPONSE_SENT
A final response was sent on a received PRACK.
RvSipCallLegTranscState

DESCRIPTION
Indicates the state of a general transaction, such as information that belongs to a call-leg.

SYNTAX
typedef enum{
    RVSIP_CALL_LEG_TRANSC_STATE_UNDEFINED = -1,
    RVSIP_CALL_LEG_TRANSC_STATE_IDLE,
    RVSIP_CALL_LEG_TRANSC_STATE_SERVER_GEN_REQUEST_RCVD,
    RVSIP_CALL_LEG_TRANSC_STATE_SERVER_GEN_FINAL_RESPONSE_SENT,
    RVSIP_CALL_LEG_TRANSC_STATE_CLIENT_GEN_REQUEST_SENT,
    RVSIP_CALL_LEG_TRANSC_STATE_CLIENT_GEN_PROCEEDING,
    RVSIP_CALL_LEG_TRANSC_STATE_TERMINATED,
    RVSIP_CALL_LEG_TRANSC_STATE_CLIENT_GEN_FINAL_RESPONSE_RCVD,
    RVSIP_CALL_LEG_TRANSC_STATE_CLIENT_MSG_SEND_FAILURE
}RvSipCallLegTranscState;

PARAMETERS
RVSIP_CALL_LEG_TRANSC_STATE_UNDEFINED
No defined state.

RVSIP_CALL_LEG_TRANSC_STATE_IDLE
The transaction state is IDLE.

RVSIP_CALL_LEG_TRANSC_STATE_SERVER_GEN_REQUEST_RCVD
The call-leg received a general request.

RVSIP_CALL_LEG_TRANSC_STATE_SERVER_GEN_FINAL_RESPONSE_SENT
A final response to the general request inside a call-leg was sent.

RVSIP_CALL_LEG_TRANSC_STATE_CLIENT_GEN_REQUEST_SENT
A general request inside a call-leg was sent.
RVSIP_CALL_LEG_TRANSC_STATE_CLIENT_GEN_PROCEEDING
The call-leg received a provisional response to the general request.

RVSIP_CALL_LEG_TRANSC_STATE_TERMINATED
The general transaction inside the call-leg is terminated.

RVSIP_CALL_LEG_TRANSC_STATE_CLIENT_GEN_FINAL_RESPONSE_RCVD
The call-leg received a final response to the general request.

RVSIP_CALL_LEG_TRANSC_STATE_CLIENT_MSG_SEND_FAILURE
The call-leg failed to send a general request due to timeout, a 503 response, or a network error.
RvSipCallLegByeState

DESCRIPTION
Indicates the state of a received BYE request.

SYNTAX
typedef enum{
    RVSIP_CALL_LEG_BYE_STATE_UNDEFINED = -1,
    RVSIP_CALL_LEG_BYE_STATE_IDLE,
    RVSIP_CALL_LEG_BYE_STATE_REQUEST_RCVD,
    RVSIP_CALL_LEG_BYE_STATE_RESPONSE_SENT,
    RVSIP_CALL_LEG_BYE_STATE_TERMINATED,
    RVSIP_CALL_LEG_BYE_STATE_DETACHED
}RvSipCallLegByeState;

PARAMETERS

RVSIP_CALL_LEG_BYE_STATE_UNDEFINED
No defined state.

RVSIP_CALL_LEG_BYE_STATE_IDLE
There is no active BYE transaction inside a call-leg.

RVSIP_CALL_LEG_BYE_STATE_REQUEST_RCVD
The call-leg received BYE request.

RVSIP_CALL_LEG_BYE_STATE_RESPONSE_SENT
A final response was sent to the BYE transaction inside a call-leg.

RVSIP_CALL_LEG_BYE_STATE_TERMINATED
The BYE transaction inside a call-leg is terminated.

RVSIP_CALL_LEG_BYE_STATE_DETACHED
The call-leg is about to detach from the transaction and there will be no more states available for this transaction.
**RvSipCallLegReplacesStatus**

**DESCRIPTION**
Indicates in which type of header the Replaces option tag appeared in an incoming request. You can get the Replaces status by calling the `RvSipCallLegGetReplacesStatus()` function.

**SYNTAX**
```c
typedef enum{
    RVSIP_CALL_LEG_REPLACES_UNDEFINED = -1,
    RVSIP_CALL_LEG_REPLACES_SUPPORTED,
    RVSIP_CALL_LEG_REPLACES_REQUIRED
}RvSipCallLegReplacesStatus;
```

**PARAMETERS**

**RVSIP_CALL_LEG_REPLACES_UNDEFINED**
The Replaces option tag was not found in the request.

**RVSIP_CALL_LEG_REPLACES_SUPPORTED**
The Replaces option tag was found in a Supported header.

**RVSIP_CALL_LEG_REPLACES_REQUIRED**
The Replaces option tag was found in a Require header.
RvSipCallLegReplacesReason

DESCRIPTION
Indicates why the call-leg that matches the identifiers in the Replaces header was not found. It is used in RvSipCallLegReplacesGetMatchedCallExt() function.

SYNTAX
typedef enum {
    RVSIP_CALL_LEG_REPLACES_REASON_UNDEFINED = -1,
    RVSIP_CALL_LEG_REPLACES_REASON_DIALOG_FOUND_OK,
    RVSIP_CALL_LEG_REPLACES_REASON_DIALOG_NOT_FOUND,
    RVSIP_CALL_LEG_REPLACES_REASON_FOUND_NON_INVITE_DIALOG,
    RVSIP_CALL_LEG_REPLACES_REASON_FOUND_CONFIRMED_DIALOG,
    RVSIP_CALL_LEG_REPLACES_REASON_FOUND_INCOMING_EARLY_DIALOG,
    RVSIP_CALL_LEG_REPLACES_REASON_FOUND_TERMINATED_DIALOG
} RvSipCallLegReplacesReason;

PARAMETERS

RVSIP_CALL_LEG_REPLACES_REASON_UNDEFINED
No specified reason.

RVSIP_CALL_LEG_REPLACES_REASON_DIALOG_FOUND_OK
The matched call-leg was found correctly.

RVSIP_CALL_LEG_REPLACES_REASON_DIALOG_NOT_FOUND
The matched call-leg was not found.

RVSIP_CALL_LEG_REPLACES_REASON_FOUND_NON_INVITE_DIALOG
A dialog was found with same dialog identifiers, but the dialog was not established with an INVITE request.
RVSIP_CALL_LEG_REPLACES_REASON_FOUND_CONFIRMED_DIALOG
A confirmed dialog was found, but the Replaces header contains the “early-only” parameter. Therefore, the application should reject the request with 486.

RVSIP_CALL_LEG_REPLACES_REASON_FOUND_INCOMING_EARLY_DIALOG
An incoming dialog was found in an early state (OFFERING), which is not allowed by the Replaces draft.

RVSIP_CALL_LEG_REPLACES_REASON_FOUND_TERMINATED_DIALOG
A dialog was found in the TERMINATED state.
**RvSipCallLegSessionTimerNotificationReason**

**DESCRIPTION**

The session-timer *notification* reason, which describes the reason for the session-timer *notification*.

**SYNTAX**

```c
typedef enum{
    RVSIP_CALL_LEG_SESSION_TIMER_NOTIFY_REASON_UNDEFINED = -1,
    RVSIP_CALL_LEG_SESSION_TIMER_NOTIFY_REASON_SESSION_EXPIRES,
    RVSIP_CALL_LEG_SESSION_TIMER_NOTIFY_REASON_422_RECEIVED
}RvSipCallLegSessionTimerNotificationReason;
```

**PARAMETERS**

**RVSIP_CALL_LEG_SESSION_TIMER_NOTIFY_REASON_UNDEFINED**

No specified reason.

**RVSIP_CALL_LEG_SESSION_TIMER_NOTIFY_REASON_SESSION_EXPIRES**

The session time is about to expire.

**RVSIP_CALL_LEG_SESSION_TIMER_NOTIFY_REASON_422_RECEIVED**

A 422 response was received during the session-timer.
RvSipCallLegSessionTimerNegotiationReason

DESCRIPTION
The session-timer negotiation reason, which describes the reason for the negotiation failure.

SYNTAX
typedef enum{
    RVSIP_CALL_LEG_SESSION_TIMER_NEGOTIATION_REASON_UNDEFINED = -1,
    RVSIP_CALL_LEG_SESSION_TIMER_NEGOTIATION_REASON_DEST_NOT_SUPPORTED,
    RVSIP_CALL_LEG_SESSION_TIMER_NEGOTIATION_REASON_REFRESHER_PREFERENCE_REJECT
}RvSipCallLegSessionTimerNegotiationReason;

PARAMETERS

RVSIP_CALL_LEG_SESSION_TIMER_NEGOTIATION_REASON_DEST_NOT_SUPPORTED
The remote party does not support the session-timer extension.

RVSIP_CALL_LEG_SESSION_TIMER_NEGOTIATION_REASON_REFRESHER_PREFERENCE_REJECT
The refresher preference does not match the call refresher.
Call-leg Type Definitions
RvSipCallLegSessionTimerRefresherPreference

**RvSipCallLegSessionTimerRefresherPreference**

**DESCRIPTION**
Saves the application preference for the call.

**SYNTAX**
```c
typedef enum{
    RVSIP_CALL_LEG_SESSION_TIMER_REFRESHER_NONE = -1,
    RVSIP_CALL_LEG_SESSION_TIMER_REFRESHER_LOCAL,
    RVSIP_CALL_LEG_SESSION_TIMER_REFRESHER_REMOTE,
    RVSIP_CALL_LEG_SESSION_TIMER_REFRESHER_DONT_CARE
}RvSipCallLegSessionTimerRefresherPreference;
```

**PARAMETERS**

**RVSIP_CALL_LEG_SESSION_TIMER_REFRESHER_LOCAL**
The UA wishes to be the refresher.

**RVSIP_CALL_LEG_SESSION_TIMER_REFRESHER_REMOTE**
The UA wishes the remote party to be the refresher.

**RVSIP_CALL_LEG_SESSION_TIMER_REFRESHER_DONT_CARE**
The UA does not care who will be the refresher.
RvSipCallLegHARestoreMode

DESCRIPTION
Notifies the Stack of high-availability mode when calling the RvSipCallLegRestoreOldVersionConnectedCall() function.

SYNTAX
typedef enum {
    RVSIP_CALL_LEG_H_A_RESTORE_MODE_UNDEFINED = -1,
    RVSIP_CALL_LEG_H_A_RESTORE_MODE_FROM_3_0_WITH_TLS,
    RVSIP_CALL_LEG_H_A_RESTORE_MODE_FROM_3_0_WITHOUT_TLS
}RvSipCallLegHARestoreMode;

PARAMETERS

RVSIP_CALL_LEG_H_A_RESTORE_MODE_FROM_3_0_WITH_TLS
The store is performed from a file generated by Stack version 3.0, compiled with TLS.

RVSIP_CALL_LEG_H_A_RESTORE_MODE_FROM_3_0_WITHOUT_TLS
The store is performed from a file generated by Stack version 3.0, compiled without TLS.
Call-leg Type Definitions
RvSipCallLegEvHandlers

RvSipCallLegEvHandlers

DESCRIPTION
A structure with function pointers to the callback of the module. This structure is used to set the application callbacks in the RvSipCallLegMgrSetEvHandlers() function.

SYNTAX
typedef struct{
    RvSipCallLegCreatedEv
        pfnCallLegCreatedEvHandler;
    RvSipCallLegStateChangedEv
        pfnStateChangedEvHandler;
    RvSipCallLegMsgToSendEv
        pfnMsgToSendEvHandler;
    RvSipCallLegMsgReceivedEv
        pfnMsgReceivedEvHandler;
    RvSipCallLegModifyStateChangedEv
        pfnModifyStateChangedEvHandler;
    RvSipCallLegModifyRequestRcvdEv
        pfnModifyRequestRcvdEvHandler;
    RvSipCallLegModifyResultRcvdEv
        pfnModifyResultRcvdEvHandler;
    RvSipCallLegTrancRequestRcvdEv
        pfnTranscRequestRcvdEvHandler;
    RvSipCallLegTranscResolvedEv
        pfnTranscResolvedEvHandler;
    RvSipCallLegReferStateChangedEv
        pfnReferStateChangedEvHandler;
    RvSipCallLegReferNotifyEv
        pfnReferNotifyEvHandler;
    RvSipCallLegPrackStateChangedEv
        pfnPrackStateChangedEvHandler;
    RvSipCallLegReferCancelledEv
        pfnReferCancelledEvHandler;
} RvSipCallLegEvHandlers;
```c
#ifdef RV_SIP_AUTH_ON
    RvSipCallLegAuthCredentialsFoundEv
        pfnAuthCredentialsFoundEvHandler;
    RvSipCallLegAuthCompletedEv
        pfnAuthCompletedEvHandler;
#endif /* #ifdef RV_SIP_AUTH_ON */

/* Session-timer callbacks */
RvSipCallLegSessionTimerRefreshAlertEv
    pfnSessionTimerRefreshAlertEvHandler;
RvSipCallLegSessionTimerNotificationEv
    pfnSessionTimerNotificationEvHandler;
RvSipCallLegSessionTimerNegotiationFaultEv
    pfnSessionTimerNegotiationFaultEvHandler;

/* BYE callbacks */
RvSipCallLegByeCreatedEv
    pfnByeCreatedEvHandler;
RvSipCallLegByeStateChangedEv
    pfnByeStateChangedEvHandler;
RvSipCallLegTranscCreatedEv
    pfnTranscCreatedEvHandler;
RvSipCallLegTranscStateChangedEv
    pfnTranscStateChangedEvHandler;
RvSipCallLegOtherURLAddressFoundEv
    pfnOtherURLAddressFoundEvHandler;
RvSipCallLegFinalDestResolvedEv
    pfnFinalDestResolvedEvHandler;
RvSipCallLegNestedInitialReqRcvdEv
    pfnNestedInitialReqRcvdEvHandler;
RvSipCallLegNewConnInUseEv
    pfnNewConnInUseEvHandler;
RvSipCallLegCreatedDueToForkingEv
    pfnCallLegCreatedDueToForkingEvHandler;
RvSipCallLegReInviteCreatedEv
    pfnReInviteCreatedEvHandler;
RvSipCallLegReInviteStateChangedEv
```
Call-leg Type Definitions
RvSipCallLegEvHandlers

pfnReInviteStateChangedEvHandler;
}RvSipCallLegEvHandlers;

PARAMETERS

pfnCallLegCreatedEvHandler
Notifies the application of a new call.

pfnStateChangedEvHandler
Notifies the application of a call-leg state change.

pfnMsgToSendEvHandler
Notifies the application that a message is about to be sent.

pfnMsgReceivedEvHandler
Notifies the application that a message was received.

pfnModifyRequestRcvdEvHandler
Notifies the application of a modify request.

pfnModifyResultRcvdEvHandler
Notifies the application of a modify request result.

pfnTrancRequestRcvdEvHandler
Notifies the application of an incoming request.

pfnTrancResolvedEvHandler
Notifies the application of the result of an outgoing request.

pfnReferStateChangedEvHandler
Notifies the application of a Refer state change.

pfnReferNotifyEvHandler
Notifies the application of refer-notify related events.
Call-leg API Type Definitions
RvSipCallLegEvHandlers

**pfnAuthCredentialsFoundEvHandler**
Notifies the application of new credentials for server authentication.

**pfnAuthCompletedEvHandler**
Notifies the application that the server authentication procedure was completed (with success or failure).

**RvSipCallLegSessionTimerRefreshAlertEv**
Notifies the application that the alert time (the time in which the application needs to send a re-INVITE to refresh the call) is expired.

**RvSipCallLegSessionTimerNotificationEv**
Notifies the application of session-timer related events.

**RvSipCallLegSessionTimerNegotiationFaultEv**
Notifies the application of session-timer negotiation fault events, such as refresher preference rejected.

**RvSipCallLegByeCreatedEv**
Notifies the application of a new BYE transaction.

**RvSipCallLegByeStateChangedEv**
Notifies the application of a BYE state change.

**RvSipCallLegOtherURLAddressFoundEv**
Notifies the application that another URL address was found and has to be converted to the SIP URL address.

**RvSipCallLegTranscCreatedEv**
Notifies the application of a new general transaction.

**RvSipCallLegTranscStateChangedEv**
Notifies the application of a general transaction state change.
Call-leg Type Definitions

**RvSipCallLegFinalDestResolvedEv**
Notifies the application that the final destination address of the register request was resolved.

**pfnNestedInitialReqRcvdEvHandler**
Notifies the application that a second (or more) INVITE was received on a *call-leg* that is not yet in the CONNECTED state. The application can decide how to act with this request.

**RvSipCallLegCreatedDueToForkingEv**
Notifies the application of a new *call-leg* that was created due to a forked 1xx/2xx response on INVITE.

**pfnReInviteCreatedEvHandler**
Notifies the application of a new Invite object, caused by a new Invite *transaction*.

**pfnReInviteStateChangedEvHandler**
Notifies the application of a re-Invite state change.
The Call-leg callback functions are:

- `RvSipCallLegCreatedEv()`
- `RvSipCallLegStateChangedEv()`
- `RvSipCallLegMsgToSendEv()`
- `RvSipCallLegMsgReceivedEv()`
- `RvSipCallLegReInviteCreatedEv()`
- `RvSipCallLegReInviteStateChangedEv()`
- `RvSipCallLegPrackStateChangedEv()`
- `RvSipCallLegAuthCredentialsFoundEv()`
- `RvSipCallLegAuthCompletedEv()`
- `RvSipCallLegSessionTimerRefreshAlertEv()`
- `RvSipCallLegSessionTimerNotificationEv()`
- `RvSipCallLegSessionTimerNegotiationFaultEv()`
- `RvSipCallLegByeCreatedEv()`
- `RvSipCallLegByeStateChangedEv()`
- `RvSipCallLegTranscCreatedEv()`
- `RvSipCallLegTranscStateChangedEv()`
- `RvSipCallLegOtherURLAddressFoundEv()`
- `RvSipCallLegFinalDestResolvedEv()`
- `RvSipCallLegNestedInitialReqRcvdEv()`
- `RvSipCallLegNewConnInUseEv()`
- `RvSipCallLegCreatedDueToForkingEv()`
- `RvSipCallLegProvisionalResponseRcvdEv()`
RvSipCallLegCreatedEv()

DESCRIPTION
Notifies the application that a new call-leg was created and exchanges handles with the application.

SYNTAX
typedef void (RVCALLCONV * RvSipCallLegCreatedEv)(
    IN RvSipCallLegHandle hCallLeg,
    OUT RvSipAppCallLegHandle *phAppCallLeg);

PARAMETERS

hCallLeg
The new SIP Stack call-leg handle.

phAppCallLeg
The application handle for this call-leg.

RETURN VALUES
None.
RvSipCallLegStateChangedEv()

**DESCRIPTION**

Notifies the application of a *call-leg* state change. For each state change, the new state is supplied with the reason for the state change.

**SYNTAX**

```c
typedef void (RVCALLCONV * RvSipCallLegStateChangedEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipCallLegState eState,
    IN RvSipCallLegStateChangeReason eReason);
```

**PARAMETERS**

- **hCallLeg**
  The SIP Stack *call-leg* handle.

- **hAppCallLeg**
  The application handle for this *call-leg*.

- **eState**
  The new *call-leg* state.

- **eReason**
  The reason for the state change.

**RETURN VALUES**

None.
Call-leg Callback Functions
RvSipCallLegMsgToSendEv()

RvSipCallLegMsgToSendEv()

DESCRIPTION
Notifies the application that a call-leg related outgoing message is about to be sent.

SYNTAX

typedef RvStatus (RVCALLCONV * RvSipCallLegMsgToSendEv)(
   IN RvSipCallLegHandle hCallLeg,
   IN RvSipAppCallLegHandle hAppCallLeg,
   IN RvSipMsgHandle hMsg);

PARAMETERS

hCallLeg
The SIP Stack call-leg handle.

hAppCallLeg
The application handle for this call-leg.

hMsg
The handle to the outgoing message.

RETURN VALUES
If you return a value other than RV_OK, the message will not be sent. The call-leg will behave as if the sending of the message failed.
RvSipCallLegMsgReceivedEv()

DESCRIPTION
Notifies the application that an incoming call-leg-related message has been received.

SYNTAX
typedef RvStatus (RVCALLCONV * RvSipCallLegMsgReceivedEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipMsgHandle hMsg);

PARAMETERS

hCallLeg
The new SIP Stack call-leg handle.

hAppCallLeg
The application handle for this call-leg.

hMsg
The handle to the outgoing message.

RETURN VALUES
If you return a value other than RV_OK, the incoming message will be ignored. The underlying transaction will terminate with an error and the call-leg will terminate as well.
RvSipCallLegReInviteCreatedEv()

**DESCRIPTION**
Notifies the application that a new INVITE object was created as a result of an incoming re-INVITE request. The SIP Stack exchanges handles with the application in this callback function.

**SYNTAX**
```c
typedef void(RVCALLCONV * RvSipCallLegReInviteCreatedEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipCallLegInviteHandle hReInvite,
    OUT RvSipAppCallLegInviteHandle *phAppReInvite,
    OUT RvUint16 *pResponseCode);
```

**PARAMETERS**
- **hCallLeg**
The SIP Stack call-leg handle.
- **hAppCallLeg**
The application handle for this call-leg.
- **hReInvite**
The handle to the new INVITE object.
- **phAppReInvite**
The application handle for this INVITE object.
- **pResponseCode**
If the application wants to reject the re-INVITE request, it can supply the response code here.

**RETURN VALUES**
None.
Call-leg Callback Functions

RvSipCallLegReInviteStateChangedEv()

DESCRIPTION

Notifies the application that the MODIFY state, related to a re-INVITE process, was changed. Before this callback function is called, a re-INVITE object must be created. (To create an outgoing re-INVITE, the application should call RvSipCallLegReInviteCreate(). An incoming re-INVITE object is created by the Stack, using the RvSipCallLegReInviteCreatedEv() callback function.

SYNTAX

typedef void
(RVCALLCONV *RvSipCallLegReInviteStateChangedEv)(
    IN  RvSipCallLegHandle    hCallLeg,
    IN  RvSipAppCallLegHandle hAppCallLeg,
    IN  RvSipCallLegInviteHandle hReInvite,
    IN  RvSipAppCallLegInviteHandle hAppReInvite,
    IN  RvSipCallLegModifyState   eModifyState,
    IN  RvSipCallLegStateChangeReason eReason);

PARAMETERS

hCallLeg
The SIP Stack call-leg handle.

hAppCallLeg
The application handle for this call-leg.

hReInvite
The SIP Stack re-INVITE handle.

hAppReInvite
The application handle for this re-INVITE.

eModifyState
The new re-INVITE state.
**Call-leg Callback Functions**

RvSipCallLegReInviteStateChangedEv()

**eReason**

The reason for the new state.

**RETURN VALUES**

None.
RvSipCallLegPrackStateChangedEv()

DESCRIPTION

Notifies the application of a call-leg PRACK event. The PRACK state indicates the status of a PRACK process. This callback is only a notification to the application and no response is expected for the different states.

SYNTAX

typedef void (RVCALLCONV * RvSipCallLegPrackStateChangedEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipCallLegPrackState eState,
    IN RvSipCallLegStateChangeReason eReason,
    IN RvInt16 prackResponseCode);

PARAMETERS

hCallLeg
The SIP Stack call-leg handle.

hAppCallLeg
The application handle for this call-leg.

eState
The PRACK process state.

eReason
The state reason.

prackResponseCode
If the state indicates that a response was received for a PRACK request, this parameter will include the response code.

RETURN VALUES

None.
**RvSipCallLegAuthCredentialsFoundEv()**

**DESCRIPTION**

Notifies the application that credentials were found in the Request message. This callback is called only if the application started the authentication procedure by calling `RvSipCallLegAuthBegin()`. The callback supplies the application with the Authorization header that contains these credentials. At this stage the application should use the `RvSipCallLegAuthProceed()` function. This callback also indicates whether the SIP Stack is capable of verifying the credentials that were found. Whenever the SIP Stack does not support the credentials (for example, if the algorithm is not MD5) the application can verify the credentials by itself or instruct the SIP Stack to continue to the next header.

**Note**  You must get the credentials from the given Authorization header synchronically, or you should copy the header to use it a-synchronously. When you have the credentials, you can check the validity and continue with `RvSipAuthProceed` a-synchronously.

**SYNTAX**

```c
typedef void (RVCALLCONV *
RvSipCallLegAuthCredentialsFoundEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipTranscHandle hTransc,
    IN RvSipAuthorizationHeaderHandle hAuthorization,
    IN RvBool bCredentialsSupported);
```

**PARAMETERS**

- **hCallLeg**
  The SIP Stack call-leg handle

- **hAppCallLeg**
  The application handle for this call-leg.
Call-leg Callback Functions
RvSipCallLegAuthCredentialsFoundEv()

hTransc
The handle to the transaction that received the response.

hAuthorization
The handle to the authorization handle that is found.

bCredentialsSupported
Determines whether or not the SIP Stack supports the algorithm in the Authorization header. If the SIP Stack supports the algorithm, the user has to return the password of this username from the realm database, and give it to the SIP Stack in the RvSipXXXAuthProceed() function, indicating the proceeding action as RVSIP_TRANSC_AUTH_ACTION_USE_PASSWORD. If the algorithm is not supported, to continue the procedure, the user can skip the unsupported header by calling RvSipXXXAuthProceed(), indicating the proceeding action as RVSIP_TRANSC_AUTH_ACTION_SKIP. Or, to end the authentication procedure successfully, the user can check the header by him/herself and call the RvSipXXXAuthProceed() function, indicating the proceeding action as RVSIP_TRANSC_AUTH_ACTION_SUCCESS.

RETURN VALUES
None.
Call-leg Callback Functions
RvSipCallLegAuthCompletedEv()

**RvSipCallLegAuthCompletedEv()**

**DESCRIPTION**
Called when the authentication procedure is completed. If the procedure is completed because the correct Authorization header was found, `bAuthSucceed` is RV_TRUE. If the procedure is completed because there are no more Authorization headers to check, or because user ordered to stop the search for the correct header, `bAuthSucceed` is RV_FALSE.

**SYNTAX**
```c
typedef void (RVCALLCONV * RvSipCallLegAuthCompletedEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipTranscHandle hTransc,
    IN RvBool bAuthSucceed);
```

**PARAMETERS**
- **hCallLeg**
  The SIP Stack *call-leg* handle.
- **hAppCallLeg**
  The application handle for this *call-leg*.
- **hTransc**
  The handle to the *transaction* that received the response.
- **bAuthSucceed**
  RV_TRUE if the correct Authorization header was found. Otherwise, RV_FALSE.

**RETURN VALUES**
None.
RvSipCallLegSessionTimerRefreshAlertEv()

**DESCRIPTION**

Notifies that the alert time (the time in which the application needs to send a re-INVITE or UPDATE to refresh the call) is expired. The application needs to send a refresh using the `RvSipCallLegTranscSessionTimerGeneralRefresh()` functions. (This callback is called only in manual mode).

**SYNTAX**

```c
typedef RvStatus (RVCALLCONV *
RvSipCallLegSessionTimerRefreshAlertEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg);
```

**PARAMETERS**

- **hCallLeg**
  
The SIP Stack call-leg handle.

- **hAppCallLeg**
  
The application handle for this call-leg.

**RETURN VALUES**

Returns `RvStatus`. 
Call-leg Callback Functions
RvSipCallLegSessionTimerNotificationEv()

RvSipCallLegSessionTimerNotificationEv()

DESCRIPTION

Notifies the application about events related to the session-timer feature. When the callback is called with the RVSIP_CALL_LEG_SESSION_TIMER_SESSION_EXPIRES reason, it notifies the application that the session time is about to expire. It is the responsibility of the application to decide whether to send BYE or to do something else. (This callback is called only in manual mode).

SYNTAX

typedef RvStatus (RVCALLCONV * RvSipCallLegSessionTimerNotificationEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipCallLegSessionTimerNotificationReason eReason);

PARAMETERS

hCallLeg
The SIP Stack call-leg handle.

hAppCallLeg
The application handle for this call-leg.

eReason
The reason for this notification.

RETURN VALUES

Returns RvStatus.
Call-leg Callback Functions

RvSipCallLegSessionTimerNegotiationFaultEv()

**DESCRIPTION**

Notifies the application about negotiation problems while defining the session-timer parameters for a call. When the callback is called with the RVSIP_CALLLEG_SESSION_TIMER_DEST_NOT_SUPPORTED reason, it notifies the local party that a 2xx final response was received, but the server does not support the session-timer. The application should return synchronously whether or not it wants to execute session-timer.

The session-timer mechanism can be operated as long as one of the two UAs in the call-leg supports the extension. If the application decides to operate the session-timer, that side will send the refresh. The other side will see the refreshes as repetitive re-Invites. The default behavior is to execute the session-timer mechanism for the call.

When the callback is called with the RVSIP_CALLLEG_SESSION_TIMER_APP_REFRESH_REQUEST_REJEC T reason, it notifies the application that the refresher preference did not match the call refresher. The application should return synchronously whether or not it wants to execute the session-timer. If the application decides to operate the session-timer mechanism, the refresher will be different from the application request. The default behavior is to execute the session-timer mechanism for the call.

**SYNTAX**

```c
typedef RvStatus (RVCALLCONV *
RvSipCallLegSessionTimerNegotiationFaultEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipCallLegSessionTimerNegotiationReason eReason,
    OUT RvBool *b_handleSessionTimer);
```

**PARAMETERS**

- **hCallLeg**
  The SIP Stack call-leg handle.
Call-leg Callback Functions
RvSipCallLegSessionTimerNegotiationFaultEv()

**hAppCallLeg**
The application handle for this call-leg.

**eReason**
The reason for this negotiation fault.

**b_handleSessionTimer**
RV_TRUE if the application wants to operate the session time of this call.
RV_FALSE if the application does not want to operate the session-timer of this call.

**RETURN VALUES**
Returns RvStatus.
RvSipCallLegByeCreatedEv()

DESCRIPTION

Notifies that a new BYE transaction was created and relates to the specified call-leg. The application can replace handles with the transaction and specify whether it wishes to handle the request. If so, it will be notified of transaction states and will have to respond to the request. If the application will not register to this callback, BYE will automatically be responded to with 200 OK.

SYNTAX

typedef void (RVCALLCONV * RvSipCallLegByeCreatedEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipTranscHandle hTransc,
    OUT RvSipAppTranscHandle *hAppTransc,
    OUT RvBool *bAppHandleTransc);

PARAMETERS

hCallLeg
The SIP Stack call-leg handle.

hAppCallLeg
The application handle for this call-leg.

hTransc
The handle to the BYE transaction.

hAppTransc
The application handle to the transaction.

bAppHandleTransc
RV_TRUE if the application wants to handle the request. RV_FALSE if the application wants the call-leg to handle the request.
Call-leg Callback Functions
RvSipCallLegByeCreatedEv()

RETURN VALUES
None.
RvSipCallLegByeStateChangedEv()

DESCRIPTION
Notifies the application of a state change for a BYE request that belongs to the supplied call-leg. When the state indicated that a request was received the application can use RvSipCallLegByeAccept() or RvSipCallLegByeReject() to respond the BYE request.

SYNTAX
typedef void (RVCALLCONV * RvSipCallLegByeStateChangedEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipTranscHandle hTransc,
    IN RvSipAppTranscHandle hAppTransc,
    IN RvSipCallLegByeState eByeState,
    IN RvSipCallLegStateChangeReason eReason);

PARAMETERS

hCallLeg
The SIP Stack call-leg handle.

hAppCallLeg
The application handle for this call-leg.

hTransc
The handle to the transaction.

hAppTransc
The application handle to the transaction.

eTranscState
The new state of the call-leg transaction.
Call-leg Callback Functions
RvSipCallLegByeStateChangedEv()

\textbf{eReason}

The reason for the new state.

\textbf{RETURN VALUES}

None.
RvSipCallLegTranscCreatedEv()

DESCRIPTION

Notifies that a new general transaction other than BYE was created and relates to the specified call-leg. The application can replace handles with the transaction and specify whether it wants to handle the request. If so, it will be notified of the transaction states and will have to respond to the request.

SYNTAX

typedef void (RVCALLCONV * RvSipCallLegTranscCreatedEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipTranscHandle hTransc,
    OUT RvSipAppTranscHandle *hAppTransc,
    OUT RvBool *bAppHandleTransc);

PARAMETERS

hCallLeg
The SIP Stack call-leg handle.

hAppCallLeg
The application handle for this call-leg.

hTransc
The handle to the new transaction.

hAppTransc
The application handle to the transaction.

bAppHandleTransc
RV_TRUE if the application wants to handle the request. RV_FALSE is the application wants the call-leg to handle the request.
Call-leg Callback Functions
RvSipCallLegTranscCreatedEv()

**RETURN VALUES**

None.
**RvSipCallLegTranscStateChangedEv()**

**DESCRIPTION**

Notifies the application of a state change for a general transaction that belongs to the supplied call-leg. When the state indicated that a request was received, the application can call `RvSipCallLegTranscResponse()` and respond to the request. If the application does not want to handle the request, it should set the `bHandleRequest` to RV_FALSE. In this case, the Stack will either handle the transaction by itself or respond with a 501 not implemented. When the state indicates that a response was received, the `responseCode` parameter will hold the response code.

**SYNTAX**

```c
typedef void
(RVCALLCONV * RvSipCallLegTranscStateChangedEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipTranscHandle hTransc,
    IN RvSipAppTranscHandle hAppTransc,
    IN RvSipCallLegTranscState eTranscState,
    IN RvSipTransactionStateChangeReason eReason);
```

**PARAMETERS**

- **hCallLeg**
  
The SIP Stack call-leg handle.

- **hAppCallLeg**
  
The application handle for this call-leg.

- **hTransc**
  
The handle to the transaction.

- **hAppTransc**
  
The application handle to the transaction.
Call-leg Callback Functions
RvSipCallLegTranscStateChangedEv()

\textbf{eTranscState}

The new state of the \textit{call-leg transaction}.

\textbf{eReason}

The reason for the new state.

**RETURN VALUES**

None.
RvSipCallLegOtherURLAddressFoundEv()

DESCRIPTION

Notifies the application that a message needs to be sent and the destination address is a URL type that is currently not supported by the SIP Stack. The URL has to be converted to a SIP URL for the message to be sent.

SYNTAX

typedef RvStatus
    (RVCALLCONV *RvSipCallLegOtherURLAddressFoundEv)(
        IN RvSipCallLegHandle hCallLeg,
        IN RvSipAppCallLegHandle hAppCallLeg,
        IN RvSipTranscHandle hTransc,
        IN RvSipAppTranscHandle hAppTransc,
        IN RvSipMsgHandle hMsg,
        IN RvSipAddressHandle hAddress,
        OUT RvSipAddressHandle hSipURLAddress,
        OUT RvBool *bAddressResolved);

PARAMETERS

hCallLeg
The SIP Stack call-leg handle.

hAppCallLeg
The application handle for this call-leg.

hTransc
The transaction handle.

hAppTransc
The application handle for this transaction.

hMsg
The message that includes the other URL address.
Call-leg Callback Functions
RvSipCallLegOtherURLAddressFoundEv()

**hAddress**
The handle to the unsupported address to be converted.

**hSipURLAddress**
The handle to the SIP URL address. This is an empty address object that the application should fill.

**bAddressResolved**
Indicates whether or not the SIP URL address was filled.

**RETURN VALUES**
Returns `RvStatus`. 
Call-leg Callback Functions

RvSipCallLegFinalDestResolvedEv()

DESCRIPTION

Notifies the application that the call-leg 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. When this callback is called, the application can query the transaction about the destination address using the RvSipTransactionGetCurrentDestAddress() function. If the application wishes, it can update the “sent-by” part of the top-most Via. The application must not update the branch parameter.

SYNTAX

typedef RvStatus
(RVCALLCONV * RvSipCallLegFinalDestResolvedEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipTranscHandle hTransc,
    IN RvSipAppTranscHandle hAppTransc,
    IN RvSipMsgHandle hMsgToSend);

PARAMETERS

hCallLeg
The SIP Stack call-leg handle.

hAppCallLeg
The application handle for this call-leg.

hTransc
The transaction handle.

hAppTransc
The application handle for this transaction. For INVITE/BYE, this parameter is NULL.
Call-leg Callback Functions
RvSipCallLegFinalDestResolvedEv()

**hMsgToSend**

The handle to the outgoing message.

**RETURN VALUES**

Returns RvStatus.

If the application returns a value other than RV_OK, the message will not be sent. The *transaction* will terminate with an error. The *call-leg* will also terminate unless the *transaction* was initiated by the application (for example, INFO).
**RvSipCallLegNestedInitialReqRcvdEv()**

**DESCRIPTION**

Notifies the application that a nested (second or more) INVITE request was received on a *call-leg* that is not yet connected. (The second INVITE request has a different *cseq* value, so it was not rejected by the Transaction layer according to the *transaction* merging rules). When this callback is called, the application can decide whether or not to order the SIP Stack to create a new *call-leg* for this nested request. If not, the SIP Stack will reject the request.

**SYNTAX**

```c
typedef void
(RVCALLCONV * RvSipCallLegNestedInitialReqRcvdEv)(
    IN RvSipCallLegHandle hExistCallLeg,
    IN RvSipAppCallLegHandle hExistAppCallLeg,
    IN RvSipMsgHandle hRcvdMsg,
    OUT RvBool *pbCreateCallLeg);
```

**PARAMETERS**

- **hExistCallLeg**
  The existing *call-leg* handle.

- **hExistAppCallLeg**
  The application handle for the existing *call-leg*.

- **hRcvdMsg**
  The handle to the nested INVITE message.

- **pbCreateCallLeg**
  RV_TRUE causes the creation of a new *call-leg*. RV_FALSE causes the rejection of the nested INVITE request.
Call-leg Callback Functions
RvSipCallLegNestedInitialReqRcvdEv()

**RETURN VALUES**

Returns RvStatus. If the application returns a value other than RV_OK, no further message will be sent.
**RvSipCallLegNewConnInUseEv()**

**DESCRIPTION**
Notifies the application that the call-leg is now using a new connection. The connection can be a new connection or a suitable one that was found in the hash.

**SYNTAX**
```c
typedef RvStatus (RVCALLCONV *RvSipCallLegNewConnInUseEv) (  
    IN RvSipCallLegHandle hCallLeg,  
    IN RvSipAppCallLegHandle hAppCallLeg,  
    IN RvSipTransportConnectionHandle hConn,  
    IN RvBool bNewConnCreated);
```

**PARAMETERS**

**hCallLeg**
The SIP Stack call-leg handle.

**hAppCallLeg**
The application handle for this call-leg.

**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. (The returned status is ignored in the current SIP Stack version.)
Call-leg Callback Functions
RvSipCallLegCreatedDueToForkingEv()

RvSipCallLegCreatedDueToForkingEv()

DESCRIPTION
An initial INVITE request may be forked by a proxy. As a result, several 1xx and 2xx responses may be received from several UASs. The first incoming response will be mapped to the original call-leg that sent the INVITE request. Every other incoming response (with a different “to-tag” parameter), creates a new “forked call-leg”.

This callback function notifies the application that a new call-leg was created due to a forked 1xx/2xx response, and exchanges handles with the application. The application may choose to terminate the new call here, using the pbTerminate parameter. If so, the call-leg will be destructed immediately and a callback will no longer be given on this call-leg.

SYNTAX
typedef void
(RVCALLCONV * RvSipCallLegCreatedDueToForkingEv)(
   IN RvSipCallLegHandle hNewCallLeg,
   OUT RvSipAppCallLegHandle *phNewAppCallLeg,
   OUT RvBool *pbTerminate);

PARAMETERS
hNewCallLeg
The new SIP Stack call-leg handle.

phNewAppCallLeg
The application handle for this call-leg.

pbTerminated
Specifies whether or not to terminate the new call-leg immediately.

RETURN VALUES
None.
RvSipCallLegProvisionalResponseRcvdEv()

**DESCRIPTION**
Notifies the application that a provisional response was received.

**SYNTAX**

typedef void

(RVCALLCONV * RvSipCallLegProvisionalResponseRcvdEv)(
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    IN RvSipTranscHandle hTransc,
    IN RvSipAppTranscHandle hAppTransc,
    IN RvSipCallLegInviteHandle hCallInvite,
    IN RvSipAppCallLegInviteHandle hAppCallInvite,
    IN RvSipMsgHandle hRcvdMsg);

**PARAMETERS**

**hCallLeg**
The SIP Stack *call-leg* handle.

**hAppCallLeg**
The application handle for this *call-leg*.

**hTransc**
The handle to the *transaction* to which the request belongs.

**hAppTransc**
The application handle to the *transaction*.

**hCallInvite**
The handle to the CInvite object.

**hAppCallInvite**
The handle to the application CInvite object.
Call-leg Callback Functions
RvSipCallLegProvisionalResponseRcvdEv()

**hRcvdMsg**

The handle to the outgoing message.

**RETURN VALUES**

None.
This section includes the SIP Stack type definitions and configuration parameters defined in the RvSipStackTypes.h file.

This section includes:

- Handle Type Definitions
- Stack Manager Type Definitions
- Configuration Parameters
- Stack Manager Callback Functions
### Handle Type Definitions

The handle type definition is:

- `RvSipStackHandle`
- `RvSipAppStackHandle`
RvSipStackHandle

**DESCRIPTION**
Declares a handle to the *StackMgr*.

**SYNTAX**

```c
RV_DECLARE_HANDLE(RvSipStackHandle);
```

**PARAMETERS**
None.
Handle Type Definitions
RvSipAppStackHandle

---

**RvSipAppStackHandle**

**DESCRIPTION**
Declares an application handle to the *StackMgr*.

**SYNTAX**

```c
RV_DECLARE_HANDLE(RvSipAppStackHandle);
```

**PARAMETERS**
None.
The Stack Manager type definitions are:

- RvSipStackModule
- RvSipStackCoreLogFiltersCfg
- RvSipStackAdsLogFiltersCfg
- RvSipStackStatistics
Stack Manager Type Definitions
RvSipStackModule

RvSipStackModule

DESCRIPTION
Defines the different modules of the C SIP Stack, including the SIP Stack, Core and ADS modules. The different values are used mainly for defining the logging filters of various components in the C SIP Stack. The RVSIP_CORE value is used to set logging filters to all the Core modules together. The RVSIP_ADS value is used to set logging filters to all ADS modules together.

SYNTAX

typedef enum {
    RVSIP_CALL,
    RVSIP_TRANSACTION,
    RVSIP_MESSAGE,
    RVSIP_TRANSPORT,
    RVSIP_PARSER,
    RVSIP_STACK,
    RVSIP_MSGBUILDER,
    RVSIP_AUTHENTICATOR,
    RVSIP_TRANSCLIENT,
    RVSIP_REGCLIENT,
    RVSIP_SUBSCRIPTION,
    RVSIP_COMPARTMENT, /*Not Supported*/
    RVSIP_RESOLVER,
    RVSIP_TRANSMITTER,
    RVSIP_SECURITY,
    RVSIP_SEC_AGREE,
    RVSIP_ADS_RLIST,
    RVSIP_ADS_RA,
    RVSIP_ADS_RPOOL,
    RVSIP_ADS_HASH,
    RVSIP_ADS_PQUEUE,
    RVSIP_CORE_SEMAPHORE,
    RVSIP_CORE_MUTEX,
    RVSIP_CORE_LOCK,
}
Stack Manager Type Definitions
RvSipStackModule

RVSIP_CORE_MEMORY,
RVSIP_CORE_THREAD,
RVSIP_CORE_QUEUE,
RVSIP_CORE_TIMER,
RVSIP_CORE_TIMESTAMP,
RVSIP_CORE_CLOCK,
RVSIP_CORE_TM,
RVSIP_CORE_SOCKET,
RVSIP_CORE_PORT_RANGE,
RVSIP_CORE_SELECT,
RVSIP_CORE_HOST,
RVSIP_CORE_TLS,
RVSIP_CORE_ARES,
RVSIP_CORE_RCACHE,
RVSIP_CORE_EHD,
RVSIP_CORE_IMSIPSEC,
RVSIP_CORE =100, /*All core modules*/
RVSIP_ADS =101

}RvSipStackModule;
RvSipStackCoreLogFiltersCfg

DESCRIPTION

Includes an entry for each of the SIP Stack Core components. This structure can be used to configure log filters separately to each of the core components.

SYNTAX

typedef struct {
    RvInt32    coreSemaphoreLogFilters;
    RvInt32    coreMutexLogFilters;
    RvInt32    coreLockLogFilters;
    RvInt32    coreMemoryLogFilters;
    RvInt32    coreThreadLogFilters;
    RvInt32    coreQueueLogFilters;
    RvInt32    coreTimerLogFilters;
    RvInt32    coreTimestampLogFilters;
    RvInt32    coreClockLogFilters;
    RvInt32    coreTmLogFilters;
    RvInt32    coreSocketLogFilters;
    RvInt32    corePortRangeLogFilters;
    RvInt32    coreSelectLogFilters;
    RvInt32    coreHostLogFilters;
    RvInt32    coreTlsLogFilters;
    RvInt32    coreAresLogFilters;
    RvInt32    coreRcacheLogFilters;
    RvInt32    coreEhdLogFilters;
    RvInt32    coreIpsecimsLogFilters;
    RvInt32    forFutureUse[20];
} RvSipStackCoreLogFiltersCfg;
RvSipStackAdsLogFiltersCfg

**DESCRIPTION**

Includes an entry for each of the SIP Stack ADS components. This structure can be used to configure log filters separately to each of the ADS components.

**SYNTAX**

```c
typedef struct {
    RvInt32   adsRListLogFilters;
    RvInt32   adsRaLogFilters;
    RvInt32   adsRPoolLogFilters;
    RvInt32   adsHashLogFilters;
    RvInt32   adsPQueueLogFilters;
    RvInt32   forFutureUse[20];
} RvSipStackAdsLogFiltersCfg;
```
RvSipStackStatistics

DESCRIPTION
Holds the SIP Stack statistics of sent and received messages.

SYNTAX
typedef struct{
    RvUint32 rcvdINVITE;
    RvUint32 rcvdINVITERetrans;
    RvUint32 rcvdNonInviteReq;
    RvUint32 rcvdNonInviteReqRetrans;
    RvUint32 rcvdResponse;
    RvUint32 rcvdResponseRetrans;
    RvUint32 sentINVITE;
    RvUint32 sentINVITERetrans;
    RvUint32 sentNonInviteReq;
    RvUint32 sentNonInviteReqRetrans;
    RvUint32 sentResponse;
    RvUint32 sentResponseRetrans;
}RvSipStackStatistics;

PARAMETERS

rcvdINVITE
Total INVITE messages received (including retransmission).

rcvdINVITERetrans
INVITE retransmissions received.

rcvdNonInviteReq
Total non-INVITE requests received (including retransmission).

rcvdNonInviteReqRetrans
Non-INVITE retransmissions received.
Stack Manager Type Definitions

rcvdResponse
Total responses received.

rcvdResponseRetrans
Responses retransmissions received.

sentINVITE
Total INVITE messages sent (including retransmission).

sentINVITERetrans
INVITE retransmissions sent.

sentNonInviteReq
Total non-INVITE requests sent (including retransmission).

sentNonInviteReqRetrans
Non-INVITE request retransmissions sent.

sentResponse
Total responses sent.

sentResponseRetrans
Response retransmissions sent.
The configuration parameter is:
- RvSipStackCfg
**RvSipStackCfg**

**DESCRIPTION**

Contains the configuration parameters and default values of the SIP Stack. The configuration parameters of the RvSipStackCfg can be divided into two groups:

- **Group A**—parameters that the SIP Stack cannot figure by itself and the application must supply. For example, maxCallLegs.
- **Group B**—parameters that the SIP Stack can calculate according the values of other configuration parameters. For example, maxTransactions. When the application does not set the maxTransactions parameter, the SIP Stack will use the following formula: maxTransactions = maxCallLegs.

The following rules apply to parameters of the RvSipStackCfg structure:

1. If you set a parameter from group A to –1, the SIP Stack will use a default value. For example, maxCallLegs will be set to 10.
2. If you set a parameter from group B to –1, the SIP Stack will calculate the value of this parameter using the values of other parameters.
3. If you set a parameter from group A to an invalid value, rule 1 will apply.
4. If you set a parameter from group B to an invalid value, rule 2 will apply.

**Note** For a detailed description of the configuration parameters, see the **Configuration** chapter in the *HP-UX C SIP Stack Programmer’s Guide*.

**SYNTAX**

```c
typedef struct {
    #ifndef RV_SIP_PRIMITIVES
    RvInt32 maxCallLegs;
    #endif /*RV_SIP_PRIMITIVES */
    RvInt32 maxTransactions;
    #ifdef RV_SIP_PRIMITIVES
    RvInt32 maxRegClients;
    #endif /*RV_SIP_PRIMITIVES */
} RvSipStackCfg;
```
Configuration Parameters
RvSipStackCfg

```c
#ifndef /*RV_SIP_PRIMITIVES */
RvInt32 messagePoolNumofPages;
RvInt32 messagePoolPageSize;
RvInt32 generalPoolNumofPages;
RvInt32 generalPoolPageSize;
RvInt32 sendReceiveBufferSize;
RvChar localUdpAddress[48];
RvUint16 localUdpPort;
RvChar outboundProxyIpAddress[48];
RvInt16 outboundProxyPort;
RvInt32 retransmissionT1;
RvInt32 retransmissionT2;
RvInt32 generalLingerTimer;
RvInt32 inviteLingerTimer;
RvInt32 provisionalTimer;
RvInt32 defaultLogFilters;
RvInt32 coreLogFilters;
RvInt32 msgLogFilters;
RvInt32 transportLogFilters;
RvInt32 transactionLogFilters;
RvInt32 callLogFilters;
RvInt32 parserLogFilters;
RvInt32 stackLogFilters;
RvInt32 msgBuilderLogFilters;
#ifdef RV_SIP_AUTH_ON
RvInt32 authenticatorLogFilters;
#endif /* #ifdef RV_SIP_AUTH_ON */
RvInt32 regClientLogFilters;

/*New parameters for version 2.0*/
RvSipStackPrintLogEntryEv pfnPrintLogEntryEvHandler;
void* logContext;
RvChar localTcpAddress[48];
RvUint16 localTcpPort;
RvInt32 maxConnections;
#endif /*RV_SIP_PRIMITIVES */
```
RvBool tcpEnabled;
#ifndef RV_SIP_PRIMITIVES
RvChar* supportedExtensionList;
RvBool rejectUnsupportedExtensions;
#endif /*RV_SIP_PRIMITIVES */
RvBool manualAckOn2xx;
RvBool isProxy;
RvInt32 proxy2xxRcvdTimer;
RvInt32 proxy2xxSentTimer;
RvChar** localUdpAddresses;
RvChar** localTcpAddresses;
RvUint16* localUdpPorts;
RvUint16* localTcpPorts;
RvInt32 numOfExtraUdpAddresses;
RvInt32 numOfExtraTcpAddresses;

/*New parameters for version 2.1*/
#ifdef RV_SIP_AUTH_ON
RvBool enableServerAuth;
#endif /* #ifdef RV_SIP_AUTH_ON */
#ifndef RV_SIP_PRIMITIVES
RvBool addSupportedListToMsg;
#endif /*RV_SIP_PRIMITIVES */
RvBool enableInviteProceedingTimeoutState;
RvInt32 retransmissionT4;
RvInt32 cancelGeneralNoResponseTimer;
RvInt32 cancelInviteNoResponseTimer;
RvInt32 generalRequestTimeoutTimer;

/*New parameters for version 2.2*/
RvUint32 maxElementsInSingleDnsList;
RvUint32 numberOfProcessingThreads;
RvInt32 processingQueueSize;
RvInt32 numOfReadBuffers;
RvUint32 processingTaskPriority;
Configuration Parameters
RvSipStackCfg

RvUint32 processingTaskStackSize;
RvInt32 subscriptionLogFilters;

#ifndef RV_SIP_PRIMITIVES
RvInt32 maxSubscriptions;
RvInt32 subsAlertTimer;
RvInt32 subsNoNotifyTimer;
RvBool subsAutoRefresh;
RvInt32 sessionExpires;
RvInt32 minSE;
#endif /*RV_SIP_PRIMITIVES */
RvBool manualBehavior;

 ресторанин new parameters for version 3.0*/
RvBool bUseRportParamInVia;
RvSipTransportPersistencyLevel ePersistencyLevel;
RvInt32 serverConnectionTimeout;
RvChar* outboundProxyHostName;
RvSipTransport eOutboundProxyTransport;

/* Tls */
RvInt32 numOfTlsAddresses;
RvChar** localTlsAddresses;
RvUint16* localTlsPorts;
RvInt32 numOfTlsEngines;
RvInt32 maxTlsSessions;
RvInt32 elementPoolNumofPages;
RvInt32 elementPageSize;
RvBool bDynamicInviteHandling;

#ifndef RV_SIP_PRIMITIVES
/*New parameters for version 3.1*/
RvBool bDisableRefer3515Behavior;
RvBool bEnableSubsForking;
RvInt32 forkedAckTrxTimeout;
RvInt32 forked1xxTimerTimeout;
#endif /*RV_SIP_PRIMITIVES*/
Configuration Parameters

RvBool manualPrack;
#endif /*RV_SIP_PRIMITIVES */
RvSipCompType eOutboundProxyCompression; (Not Supported)
RvInt32 compartmentLogFilters; (Not Supported)
RvInt32 transmitterLogFilters;
RvInt32 maxCompartments; (Not Supported)
RvInt32 sigCompTcpTimer; (Not Supported)
RvInt32 maxNumOfLocalAddresses;
RvBool bEnableForking;
RvBool bDisableMerging;
RvInt32 adsLogFilters;
RvSipStackAdsLogFiltersCfg adsFiltersCfg;
RvSipStackCoreLogFiltersCfg coreFiltersCfg;
RvSipTransportAddr* pDnsServers;
RvInt32 numOfDnsServers;
RvChar** pDnsDomains;
RvInt32 numOfDnsDomains;
RvInt32 maxDnsServers;
RvInt32 maxDnsDomains;
RvInt32 maxDnsBuffLen;
RvBool bDLAEnabled;
RvInt32 maxTransmitters;

/*New parameters for version 4.0*/
RvInt32 resolverLogFilters;
RvBool bOldInviteHandling;
RvBool bResolveTelUrls;
RvChar* strDialPlanSuffix;
RvInt32 numOfOutStreams;
RvInt32 numOfInStreams;
RvInt32 avgNumOfMultihomingAddresses;
RvBool bIgnoreLocalAddresses;
}RvSipStackCfg;
The Stack Manager callback function is:

 a. RvSipStackPrintLogEntryEv()
Stack Manager Callback Functions

RvSipStackPrintLogEntryEv()  

DESCRIPTION

Notifies the application each time a line should be printed to the log. The application can decide whether to print the line to the screen, file or other output device. You set this callback in the RvSipStackCfg structure before initializing the SIP Stack. If you do not implement this function, a default logging is used and the line is written to the SipLog.txt file.

SYNTAX

typedef void (RVCALLCONV * RvSipStackPrintLogEntryEv)(
    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 log. The text is formatted as follows:
<filer> - <module> - <message>
For example,
INFO - STACK - Stack was constructed successfully

RETURN VALUES

None.
Stack Manager Callback Functions
RvSipStackPrintLogEntryEv()
This section includes the SIP Stack Transaction type definitions and callback functions defined in the RvSipTransactionTypes.h header file.

This section includes:

- Handle Type Definitions
- Transaction Type Definitions
- Transaction Callback Functions
- Transaction Manager Callback Functions
Handle Type Definitions

The handle type definitions are:

- RvSipTranscMgrHandle
- RvSipTranscHandle
- RvSipTranscOwnerHandle
- RvSipAppTranscHandle
RvSipTranscMgrHandle

**DESCRIPTION**

The declaration of a handle to a TransactionMgr instance. The TransactionMgr manages all the transactions and the handle is needed in all manager-related API functions, such as the creation of new transactions.

**SYNTAX**

```c
RV_DECLARE_HANDLE(RvSipTranscMgrHandle);
```

**PARAMETERS**

None.

**RETURN VALUES**

None.
RvSipTranscHandle

DESCRIPTION
The declaration of a transaction handle. The transaction handle is required by all Transaction API functions, and is used to reference the correct transaction.

SYNTAX
RV_DECLARE_HANDLE(RvSipTranscHandle);

PARAMETERS
None.

RETURN VALUES
None.
RvSipTranscOwnerHandle

DESCRIPTION
The declaration of the application handle to a transaction. This handle is used by the application in order to associate SIP Stack transactions with the application transaction. The application gives the application handle when a new transaction is created. The SIP Stack will give this handle back to the application in every callback function.

SYNTAX
RV_DECLARE_HANDLE(RvSipTranscOwnerHandle);

PARAMETERS
None.

RETURN VALUES
None.
RvSipAppTranscHandle

**DEFINITION**

The declaration of application handle to a transaction. This handle is used by the application to associate SIP Stack transactions with application transactions. This handle acts as a context that the application can hold in the transaction. The application can set and get this handle with the RvSipTransactionGetAppHandle() and RvSipTransactionSetAppHandle() API functions. This handle is not supplied in the transaction callback functions. It is used for call-leg transactions.

**SYNTAX**

RV_DECLARE_HANDLE(RvSipAppTranscHandle);

**PARAMETERS**

None.

**RETURN VALUES**

None.
The Transaction type definitions are:

- RvSipTransactionState
- RvSipTransactionStateChangeReason
- RvSipTransaction100RelStatus
- RvSipTransactionAuthAction
- RvSipTransactionEvHandlers
- RvSipTransactionMgrEvHandlers
Transaction Type Definitions
RvSipTransactionState

RvSipTransactionState

DESCRIPTION

Represents the state of the transaction. The RvSipTransactionStateChangedEv() callback reports transaction state changes and state change reasons. The state change reason indicates why the transaction reached the new state. The transaction associates with the states enumerated below.

Note For a detailed description of the transaction states, see the Working with Transactions chapter in the HP-UX C SIP Stack Programmer's Guide.

SYNTAX

typedef enum{
   RVSIP_TRANSC_STATE_UNDEFINED = -1,
   RVSIP_TRANSC_STATE_IDLE,
   RVSIP_TRANSC_STATE_SERVER_GEN_REQUEST_RCVD,
   RVSIP_TRANSC_STATE_SERVER_GEN_FINAL_RESPONSE_SENT,
   RVSIP_TRANSC_STATE_CLIENT_GEN_REQUEST_SENT,
   RVSIP_TRANSC_STATE_CLIENT_GEN_PROCEEDING,
   RVSIP_TRANSC_STATE_CLIENT_INVITE_CALLING,
   RVSIP_TRANSC_STATE_CLIENT_INVITE_PROCEEDING,
   RVSIP_TRANSC_STATE_CLIENT_INVITE_FINAL_RESPONSE_RCVD,
   RVSIP_TRANSC_STATE_CLIENT_INVITE_ACK_SENT,
   RVSIP_TRANSC_STATE_SERVER_INVITE_REQUEST_RCVD,
   RVSIP_TRANSC_STATE_SERVER_INVITE_FINAL_RESPONSE_SENT,
   RVSIP_TRANSC_STATE_TERMINATED,
   RVSIP_TRANSC_STATE_SERVER_PRACK_FINAL_RESPONSE_SENT,
   RVSIP_TRANSC_STATE_SERVER_GEN_FINAL_RESPONSE_SENT,
   RVSIP_TRANSC_STATE_SERVER_INVITE_PRACK_COMPLETED = 13,
   RVSIP_TRANSC_STATE_SERVER_INVITE_REL_PROV_RESPONSE_SENT,
   RVSIP_TRANSC_STATE_CLIENT_INVITE_PROXY_2XX_RESPONSE_RCVD,
   RVSIP_TRANSC_STATE_SERVER_INVITE_PROXY_2XX_RESPONSE_SENT,
   RVSIP_TRANSC_STATE_SERVER_CANCEL_REQUEST_RCVD,
   RVSIP_TRANSC_STATE_CLIENT_INVITE_CANCELLING,
}
RvSipTransactionState

RVSIP_TRANSC_STATE_CLIENT_GEN_CANCELING,
RVSIP_TRANSC_STATE_CLIENT_CANCEL_SENT,
RVSIP_TRANSC_STATE_CLIENT_CANCEL_PROCEEDING,
RVSIP_TRANSC_STATE_SERVER_CANCEL_FINAL_RESPONSE_SENT,
RVSIP_TRANSC_STATE_CLIENT_GEN_FINAL_RESPONSE_RCVD,
RVSIP_TRANSC_STATE_CLIENT_INVITE_PROCEEDING_TIMEOUT,
RVSIP_TRANSC_STATE_SERVER_INVITE_ACK_RCVD,
RVSIP_TRANSC_STATE_CLIENT_CANCEL_FINAL_RESPONSE_RCVD,
RVSIP_TRANSC_STATE_CLIENT_MSG_SEND_FAILURE

}RvSipTransactionState;
RvSipTransactionStateChangeReason

DESCRIPTION

The reason for a state change. A transaction has an inner state. The state changes according to the state machine defined for the transactions. Each state change is explained by one of the reasons enumerated below.

SYNTAX

typedef enum{
    RVSIP_TRANSC_REASON_ERROR = -2,
    RVSIP_TRANSC_REASON_UNDEFINED,
    RVSIP_TRANSC_REASON_USER_COMMAND,
    RVSIP_TRANSC_REASON_TIME_OUT,
    RVSIP_TRANSC_REASON_RAN_OUT_OF_TIMERS,
    RVSIP_TRANSC_REASON_NORMAL_TERMINATION,
    RVSIP_TRANSC_REASON_TRANSACTION_MANAGER_COMMAND,
    RVSIP_TRANSC_REASON_REQUEST_RECEIVED,
    RVSIP_TRANSC_REASON_ACK_RECEIVED,
    RVSIP_TRANSC_REASON_PROVISIONAL_RESPONSE_RECEIVED,
    RVSIP_TRANSC_REASON_RESPONSE_SUCCESSFUL_RECVD,
    RVSIP_TRANSC_REASON_RESPONSE_REDIRRECVD,
    RVSIP_TRANSC_REASON_RESPONSE_REQUEST_FAILURE_RECVD,
    RVSIP_TRANSC_REASON_RESPONSE_SERVER_FAILURE_RECVD,
    RVSIP_TRANSC_REASON_RESPONSE_GLOBAL_FAILURE_RECVD,
    RVSIP_TRANSC_REASON_TRANSACTION_CANCELED,
    RVSIP_TRANSC_REASON_RELIABLE_PROVISIONAL_RCVD,
    RVSIP_TRANSC_REASON_TRANSACTION_COMMAND,
    RVSIP_TRANSC_REASON_UNSUPPORTED_OPTION,
    RVSIP_TRANSC_REASON_NETWORK_ERROR,
    RVSIP_TRANSC_REASON_CONTINUE_DNS,
    RVSIP_TRANSC_REASON_CALL_LEG_MANAGER_COMMAND,
    RVSIP_TRANSC_REASON_DETACH_OWNER,
    RVSIP_TRANSC_REASON_REL_PROV_RESP_TIME_OUT,
    RVSIP_TRANSC_REASON_OUT_OF_RESOURCES
}
Transaction Type Definitions
RvSipTransactionStateChangeReason

}RvSipTransactionStateChangeReason;
**RvSipTransaction100RelStatus**

**DESCRIPTION**
Indicates in which type of header the 100rel option tag appeared in an incoming request. You can get the 100 rel status by calling the function, `RvSipTransactionGet100RelStatus()`.

**SYNTAX**
```c
typedef enum{
    RVSIP_TRANSC_100_REL_UNDEFINED = -1,
    RVSIP_TRANSC_100_REL_SUPPORTED,
    RVSIP_TRANSC_100_REL_REQUIRED
}RvSipTransaction100RelStatus;
```

**PARAMETERS**

**RVSIP_TRANSC_100_REL_UNDEFINED**
The 100rel was not found in the request.

**RVSIP_TRANSC_100_REL_SUPPORTED**
The 100rel was found in a Supported header.

**RVSIP_TRANSC_100_REL_REQUIRED**
The 100rel was found in a Require header.
RvSipTransactionAuthAction

DESCRIPTION
Indicates to the RvSipTransactionAuthProceed() function with which action to proceed with the authentication process.

SYNTAX
typedef enum{
    RVSIP_TRANSC_AUTH_ACTION_UNDEFINED = -1,
    RVSIP_TRANSC_AUTH_ACTION_USE_PASSWORD,
    RVSIP_TRANSC_AUTH_ACTION_SUCCESS,
    RVSIP_TRANSC_AUTH_ACTION_FAILURE,
    RVSIP_TRANSC_AUTH_ACTION_SKIP
}RvSipTransactionAuthAction;

PARAMETERS

RVSIP_TRANSC_AUTH_ACTION_USE_PASSWORD
Orders to check the given Authorization header, using the given password.

RVSIP_TRANSC_AUTH_ACTION_SUCCESS
The user had checked the Authorization header and it is correct. (This will cause AuthCompletedEv to be called, with Success status).

RVSIP_TRANSC_AUTH_ACTION_FAILURE
The user wants to stop the loop that searches for Authorization headers. (This will cause AuthCompletedEv to be called, with Failure status).

RVSIP_TRANSC_AUTH_ACTION_SKIP
Orders to skip the given header, and continue the authentication procedure with the next header (if it exists). (This will cause AuthCredentialFoundEv to be called, or AuthCompletedEv (with Failure status) if there are no more Authorization headers.)
RvSipTransactionEvHandlers

DESCRIPTION
A structure with function pointers to the callback of the module. This structure
is used to set the application callbacks in the
RvSipTransactionMgrSetEvHandlers() function.

SYNTAX
typedef struct{
    RvSipTransactionCreatedEv
        pfnEvTransactionCreated;
    RvSipTransactionStateChangedEv
        pfnEvStateChanged;
    RvSipTransactionMsgToSendEv
        pfnEvMsgToSend;
    RvSipTransactionMsgReceivedEv
        pfnEvMsgReceived;
    RvSipTransactionInternalClientCreatedEv
        pfnEvInternalTranscCreated;
    RvSipTransactionCancelledEv
        pfnEvTranscCancelled;
#ifdef RV_SIP_AUTH_ON
    RvSipTransactionAuthCredentialsFoundEv
        pfnEvAuthCredentialsFound;
    RvSipTransactionAuthCompletedEv
        pfnEvAuthCompleted;
#endif /* #ifdef RV_SIP_AUTH_ON */
#ifndef RV_SIP_PRIMITIVES
    RvSipTransactionOpenCallLegEv
        pfnEvOpenCallLeg;
#endif /* #ifndef RV_SIP_PRIMITIVES */
#endif /* #ifdef RV_SIP_PRIMITIVES */
    RvSipTransactionOtherURLAddressFoundEv
        pfnEvOtherURLAddressFound;
    RvSipTransactionFinalDestResolvedEv
        pfnEvFinalDestResolved;
} RvSipTransactionEvHandlers;
RvSipTransactionNewConnInUseEv
   pfnEvNewConnInUse;
#elifdef RV_SIP_PRIMITIVES
   RvSipTransactionMgrForkedInviteResponseEv
   pfnEvForkedInviteRespRcvd;
#endif /*#ifdef RV_SIP_PRIMITIVES*/
}RvSipTransactionEvHandlers;

PARAMETERS

pfnEvTransactionCreated
Notifies that a new server transaction was created.

pfnEvStateChanged
Notifies of a transaction state change.

pfnEvMsgToSend
Notifies that a message is about to be sent.

pfnEvMsgReceived
Notifies that a message was received.

pfnEvInternalTranscCreated
Notifies that an internal client transaction was created. This callback is called for CANCEL and PRACK transactions that are created automatically by the SIP Stack.

pfnEvTranscCancelled
Notifies that a transaction was cancelled by an incoming CANCEL request.

pfnEvAuthCredentialsFound
Supplies an Authorization header to the application so that an authentication check will be performed for an incoming request.

pfnEvAuthCompleted
Notifies that the authentication procedure is completed.
**Transaction Type Definitions**

**RvSipTransactionEvHandlers**

**pfnEvOpenCallLeg**

Notifies the application that a request that is suitable for opening a dialog was received (INVITE/REFER/SUBSCRIBE—with no to-tag parameter). The application should decide whether to open a new dialog or handle the request above the Transaction layer.

**pfnEvOtherURLAddressFound**

Notifies the application that the destination address of an outgoing message is not a SIP URL. The application must convert the address to a SIP URL address for the message to be sent.

**pfnEvFinalDestResolved**

Notifies that the transaction is about to send a message after the destination address was resolved. This callback supplies the final message object. The application can use the Transaction API to get the destination address.

**pfnEvNewConnInUse**

Notifies that the transaction is now using a new connection. The connection can be a totally new connection or a suitable connection that was found in the hash.

**pfnEvForkedInviteRespRcvd**

Notifies that the TransactionMgr received a new 1xx/2xx response for INVITE, and the response was not related to any transaction. (This is usually the case when an INVITE request was forked by a proxy to several UAS, and each one of them returned 1xx and 2xx).
RvSipTransactionMgrEvHandlers

DESCRIPTION

A structure with function pointers to the TransactionMgr callback. TransactionMgr callbacks are not related to a specific transaction. You set these event handlers using the RvSipTransactionMgrSetMgrEvHandlers() function.

SYNTAX

typedef struct{
    RvSipTranscMgrOutOfContextMsgRcvdEv pfnEvOutOfContextMsg;
    RvSipTranscMgrNewRequestRcvdEv pfnEvNewRequestRcvd;
}RvSipTransactionMgrEvHandlers;

PARAMETERS

pfnEvOutOfContextMsg

Notifies the application about receiving an Out of Context message. Proxies will be notified of all messages. User Agents will be notified on ACK, 1xx and 2xx messages that do not match any transaction.

pfnEvNewRequestRcvd

Notifies the application when a new request is received. The application has to determine whether or not to open a new transaction for the request. User Agents will usually not register to this callback.
The Transaction callback functions are:

- RvSipTransactionCreateEv()
- RvSipTransactionStateChangedEv()
- RvSipTransactionMsgToSendEv()
- RvSipTransactionMsgReceivedEv()
- RvSipTransactionInternalClientCreatedEv()
- RvSipTransactionCancelledEv()
- RvSipTransactionAuthCredentialsFoundEv()
- RvSipTransactionAuthCompletedEv()
- RvSipTransactionOpenCallLegEv()
- RvSipTransactionOtherURLAddressFoundEv()
- RvSipTransactionFinalDestResolvedEv()
- RvSipTransactionNewConnInUseEv()
- RvSipTransactionMgrForkedInviteResponseEv()
RvSipTransactionCreateEv()

DESCRIPTION
Notifies the application that a new server transaction has been created. The newly created transaction always assumes the IDLE state. You should decide whether you wish to handle this transaction. If so, your application can exchange handles with the SIP Stack using this callback.

SYNTAX
typedef void (RVCALLCONV * RvSipTransactionCreatedEv)(
    IN RvSipTranscHandle hTransc,
    IN void *pAppContext,
    OUT RvSipTranscOwnerHandle *hAppTransc,
    OUT RvBool *b_handleTransc);

PARAMETERS
hTransc
The new SIP Stack transaction handle.

pAppContext
The application context which you supplied.

hAppTransc
The application handle for the transaction.

b_handleTransc
Indicates whether the application wishes to handle the transaction—RV_TRUE. If set to RV_FALSE, the SIP Stack will handle the transaction by itself. The normal behavior will be rejecting the request with a status code of 501 (Not Implemented).

RETURN VALUES
None.
RvSipTransactionStateChangedEv()

DESCRIPTION

Notifies the application of a transaction state change and the associated state change reason. This function is probably the most useful of the events that the SIP transaction reports. When you receive notifications of SIP transaction state changes, your application can act upon the state. For example, upon receipt of a SERVER_GENERAL_REQUEST_RECEIVED state notification, your application can respond with a desired response code.

SYNTAX

typedef void (RVCALLCONV * RvSipTransactionStateChangedEv)(
    IN RvSipTranscHandle hTransc,
    IN RvSipTranscOwnerHandle hTranscOwner,
    IN RvSipTransactionState eState,
    IN RvSipTransactionStateChangeReason eReason);

Parameters

hTransc
The transaction handle.

hTranscOwner
The application handle for this transaction.

eState
The new transaction state.

eReason
The reason for the state change.

RETURN VALUES

None.
RvSipTransactionMsgToSendEv()

**DESCRIPTION**
Indicates that a *transaction*-related outgoing message is about to be sent.

**SYNTAX**
```c
typedef RvStatus (RVCALLCONV * RvSipTransactionMsgToSendEv)(
    IN RvSipTranscHandle hTransc,
    IN RvSipTranscOwnerHandle hTranscOwner,
    IN RvSipMsgHandle hMsgToSend);
```

**PARAMETERS**
- **hTransc**  
  The *transaction* handle.
- **hTranscOwner**  
  The application handle for this *transaction*.
- **hMsgToSend**  
  The handle to the outgoing message.

**RETURN VALUES**
If you return a value other than RV_OK, the message will not be sent.
Transaction Callback Functions
RvSipTransactionMsgReceivedEv()

RvSipTransactionMsgReceivedEv()

DESCRIPTION
Indicates that a transaction-related incoming message has been received.

SYNTAX
typedef RvStatus
  (RVCALLCONV * RvSipTransactionMsgReceivedEv)(
    IN RvSipTranscHandle hTransc,
    IN RvSipTranscOwnerHandle hTranscOwner,
    IN RvSipMsgHandle hMsgReceived);

PARAMETERS
hTransc
The transaction handle.

hTranscOwner
The application handle for this transaction.

hMsgReceived
The handle to the outgoing message.

RETURN VALUES
Returning a value other than RV_OK will cause the transaction to terminate with an error.
**RvSipTransactionInternalClientCreatedEv()**

**DESCRIPTION**

RvSipTransactionCreateEv() notifies the application that a new client transaction has been created by the SIP Stack. The newly created transaction always assumes the IDLE state. This callback is called for CANCEL and PRACK transactions that are created automatically by the SIP Stack.

**SYNTAX**

typedef void

(RVCALLCONV * RvSipTransactionInternalClientCreatedEv)(
    IN RvSipTranscHandle hTransc,
    IN void *pAppContext,
    OUT RvSipTranscOwnerHandle *hAppTransc);

**PARAMETERS**

**hTransc**

The new SIP Stack transaction handle.

**pAppContext**

The application context. You supply this context when setting the event handles.

**hAppTransc**

The application handle for the transaction.

**RETURN VALUES**

None.
**RvSipTransactionCancelledEv()**

**DESCRIPTION**
Notifies the application when a CANCEL request is received on a transaction.

**Note**  This is not the CANCEL transaction but the cancelled transaction. For example an INVITE request that was cancelled.

**SYNTAX**
```c
typedef RvStatus (RVCALLCONV * RvSipTransactionCancelledEv)(
    IN RvSipTranscHandle hTransc,
    IN RvSipTranscOwnerHandle hTranscOwner);
```

**PARAMETERS**
- **hTransc**
The transaction handle.
- **hTranscOwner**
The application handle for this transaction.

**RETURN VALUES**
Returns RvStatus.
RvSipTransactionAuthCredentialsFoundEv()

DESCRIPTION

Supplies an Authorization header for passing to the user, that will continue the authentication procedure according to the realm and username parameters in the header. To continue the procedure, use RvSipTransactionAuthProceed().

Note  You must get the credentials from the given Authorization header synchronically, or you should copy the header to use it a-synchronously. When you have the credentials, you can check the validity and continue with RvSipAuthProceed a-synchronously.

SYNTAX

typedef void
(RVDCALLCONV * RvSipTransactionAuthCredentialsFoundEv)(
    IN RvSipTranscHandle hTransc,
    IN RvSipTranscOwnerHandle hAppTransc,
    IN RvSipAuthorizationHeaderHandle hAuthorization,
    IN RvBool bCredentialsSupported);

PARAMETERS

hTransc
The SIP Stack transaction handle.

hAppTransc
The application handle for the transaction.

hAuthorization
The handle to the Authorization header that was found.

bCredentialsSupported
TRUE if supported, otherwise FALSE.
Transaction Callback Functions
RvSipTransactionAuthCredentialsFoundEv()

RETURN VALUES

None.
RvSipTransactionAuthCompletedEv()

DESCRIPTION
Notifies that the authentication procedure is completed. If the procedure is completed because a correct authorization was found, bAuthSucceed is RV_TRUE. If the procedure is completed because there are no more Authorization headers to check, or because the user ordered to stop the search for the correct header, bAuthSucceed is RV_FALSE.

SYNTAX
typedef void (RVCALLCONV * RvSipTransactionAuthCompletedEv)(
    IN RvSipTranscHandle hTransc,
    IN RvSipTranscOwnerHandle hAppTransc,
    IN RvBool bAuthSucceed);

PARAMETERS
hTransc
The SIP Stack transaction handle.

hAppTransc
The application handle for the transaction.

bAuthSucceed
RV_TRUE if the correct Authorization header was found. Otherwise, RV_FALSE.

RETURN VALUES
None.
RvSipTransactionOpenCallLegEv()

**DESCRIPTION**

When a request that is suitable for opening a dialog is received (INVITE/REFER/SUBSCRIBE—with no to-tag parameter), the Transaction layer asks the application whether to open a *call-leg* for this *transaction*. For a proxy application, the callback is called for INVITE/REFER/SUBSCRIBE methods. This function can be used by proxies that wish to handle specific requests in a *call-leg* context by themselves. For UA applications, the callback is called only for initial REFER/SUBSCRIBE methods. An application that does not want the Stack implementation that opens a new dialog for REFER and SUBSCRIBE should implement this callback.

**SYNTAX**

```c
typedef RvStatus
    (RVCALLCONV * RvSipTransactionOpenCallLegEv)(
    IN RvSipTranscHandle hTransc,
    OUT RvBool *bOpenCalled);
```

**PARAMETERS**

- **hTransc**
  The handle to the new server *transaction*.

- **bOpenCalled**
  RV_TRUE if the application wishes that the Stack will handle the *transaction* in a *call-leg* context. Otherwise, RV_FALSE.

**RETURN VALUES**

Returns *RvStatus*. (The returned status is ignored in the current SIP Stack version.)
**RvSipTransactionOtherURLAddressFoundEv()**

**DESCRIPTION**

Notifies the application that a message needs to be sent and the destination address is a URL type that is currently not supported by the stack. The URL has to be converted to a SIP URL for the message to be sent.

**SYNTAX**

```c
typedef RvStatus
        (RVCALLCONV * RvSipTransactionOtherURLAddressFoundEv)(
        IN RvSipTranscHandle hTransc,
        IN RvSipTranscOwnerHandle hAppTransc,
        IN RvSipMsgHandle hMsg,
        IN RvSipAddressHandle hAddress,
        OUT RvSipAddressHandle hSipURLAddress,
        OUT RvBool *bAddressResolved);
```

**PARAMETERS**

- **hTransc**
  The transaction handle.

- **hAppTransc**
  The application handle for this transaction.

- **hMsg**
  The message that includes the other URL address.

- **hAddress**
  The handle to the unsupported address to be converted.

- **hSipURLAddress**
  The handle to the SIP URL address. This is an empty address object that the application should fill.
Transaction Callback Functions
RvSipTransactionOtherURLAddressFoundEv()

bAddressResolved
RV_TRUE if the SIP URL address was filled. Otherwise, RV_FALSE.

RETURN VALUES
Returns RvStatus.
RvSipTransactionFinalDestResolvedEv()

DESCRIPTION

Notifies the application that the transaction is about to send a message after the destination address was resolved. This callback supplies the final message object. Changes in the message at this stage will not effect the destination address. When this callback is called, the application can query the transaction about the destination address using the RvSipTransactionGetCurrentDestAddress() function. If the application wishes, it can update the sent-by part of the top-most Via header. The application must not update the branch parameter.

SYNTAX

typedef RvStatus
    (RVCALLCONV *RvSipTransactionFinalDestResolvedEv)(
        IN RvSipTranscHandle hTransc,
        IN RvSipTranscOwnerHandle hTranscOwner,
        IN RvSipMsgHandle hMsgToSend);

PARAMETERS

hTransc

The transaction handle.

hTranscOwner

The application handle to this transaction.

hMsgToSend

The handle to the outgoing message.

RETURN VALUES

Returns RvStatus.

If the application returns a value other than RV_OK, the message will not be sent. The transaction will terminate with an error.
**Transaction Callback Functions**

RvSipTransactionNewConnInUseEv()

---

### RvSipTransactionNewConnInUseEv()

#### DESCRIPTION

Notifies the application that the transaction is now using a new connection. The connection can be a totally new one or a suitable one that was found in the hash.

#### SYNTAX

```c
typedef RvStatus
   (RVCALLCONV *RvSipTransactionNewConnInUseEv) (
    IN RvSipTranscHandle hTransc,
    IN RvSipTranscOwnerHandle hTranscOwner,
    IN RvSipTransportConnectionHandle hConn,
    IN RvBool bNewConnCreated);
```

#### PARAMETERS

- **hTransc**
  
  The SIP Stack transaction handle.

- **hTranscOwner**
  
  The owner of this transaction.

- **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. (The returned status is ignored in the current SIP Stack version.)
RvSipTransactionMgrForkedInviteResponseEv()

DESCRIPTION

Called when the TransactionMgr receives a new 1xx/2xx response for an INVITE request, and the response does not match an existing transaction. (This is usually the case when an INVITE request was forked by proxy to several UASs, and each one of them returns 1xx and 2xx.)

Note  This callback is available only if the SIP Stack is compiled with the RV_SIP_PRIMITIVES compilation flag.

SYNTAX

typedef RvStatus
   (RVCALLCONV RvSipTransactionMgrForkedInviteResponseEv)(
      IN void* hTranscMgr,
      IN RvSipMsgHandle hResponseMsg);

PARAMETERS

hTranscMgr

The TransactionMgr that received the response.

hResponseMsg

The handle to the received response message.

RETURN VALUES

Returns RvStatus. (The returned status is ignored in the current SIP Stack version.)
The Transaction Manager (TransactionMgr) callback functions are:

- RvSipTranscMgrOutOfContextMsgRcvdEv()
- RvSipTranscMgrNewRequestRcvdEv()
RvSipTranscMgrOutOfContextMsgRcvdEv()

DESCRIPTION

Called when the TransactionMgr receives a message that does not match any existing SIP Stack object. The callback is called in the following cases:

- An ACK is received and does not match any existing transaction or dialog. (UA applications have to configure the SIP Stack to work with dynamic INVITE handling to receive this callback for such ACK requests.)
- A response message is received and there is no matching client transaction.
- A CANCEL is received and there is no matching INVITE transaction to cancel.
- The application returned RV_FALSE in the RvSipTranscMgrNewRequestRcvdEv() callback, indicating that it does not wish the SIP Stack to create a new transaction for an incoming request.

SYNTAX

typedef RvStatus
    (RVCALLCONV *RvSipTranscMgrOutOfContextMsgRcvdEv)(
    IN RvSipTranscMgrHandle hTranscMgr,
    IN void* pAppMgr,
    IN RvSipMsgHandle hMsg,
    IN RvSipTransportLocalAddrHandle hLocalAddr,
    IN RvSipTransportAddr* pRcvdFromAddr,
    IN RvSipTransportAddrOptions* pOptions,
    IN RvSipTransportConnectionHandle hConn);

PARAMETERS

hTranscMgr

The TransactionMgr that received the message.

pAppMgr

The application context.
Transaction Manager Callback Functions
RvSipTranscMgrOutOfContextMsgRcvdEv()

hMsg
The received Out of Context message.

hLocalAddr
The local address on which the message was received.

pRcvdFromAddr
The address from where the message was received.

pOptions
Other identifiers of the message.

hConn
If the message was received on a specific connection, the connection handle is supplied. Otherwise this parameter is set to NULL.

RETURN VALUES
None.
RvSipTranscMgrNewRequestRcvdEv()

DESCRIPTION
Called when the TransactionMgr receives a new request. This callback will not be called in the following cases:

- If the request is a retransmission.
- If the request method is ACK.
- If a proxy application received a CANCEL that does not match any INVITE transaction.

The application should instruct the SIP Stack whether it should create a new transaction for the request or not. If you do not implement this callback, a new transaction will be created by default. Stateless proxies will usually want to prevent the SIP Stack from creating a new transaction.

SYNTAX

typedef RvStatus (RvSipTranscMgrNewRequestRcvdEv)(
    IN RvSipTranscMgrHandle hTranscMgr,
    IN void *pAppMgr,
    IN RvSipMsgHandle hMsg,
    OUT RvBool *pbCreateTransc);

PARAMETERS

hTranscMgr
The TransactionMgr that received the request.

pAppMgr
The application context. You supply this context when setting the event handles.

hMsg
The handle to the new Request message.

pbCreateTransc
Indicates whether the SIP Stack should handle the new request by creating a new transaction.
Transaction Manager Callback Functions
RvSipTranscMgrNewRequestRcvdEv()

RETURN VALUES

Returns RvStatus.

The returned status is ignored in the current SIP Stack version.
TRANSMITTER API TYPE DEFINITIONS

This section includes the SIP Stack Transmitter type definitions and callback functions defined in the \textit{RvSipTransmitterTypes.h} header file.

This section includes:

- Handle Type Definitions
- Transmitter Type Definitions
- Transmitter Callback Functions
Handle Type Definitions

HANDLE TYPE DEFINITIONS

The handle type definitions are:

- RvSipTransmitterMgrHandle
- RvSipTransmitterHandle
- RvSipAppTransmitterHandle
RvSipTransmitterMgrHandle

**DESCRIPTION**

The declaration of handle to a `TransmitterMgr` instance. The `TransmitterMgr` manages all `transmitters` and its handle is needed in all manager-related functions, such as creation of new `transmitters`.

**SYNTAX**

```c
RV_DECLARE_HANDLE(RvSipTransmitterMgrHandle);
```
RvSipTransmitterHandle

DESCRIPTION

The declaration of a transmitter handle. A transmitter handle is used to reference a transmitter.

SYNTAX

RV_DECLARE_HANDLE(RvSipTransmitterHandle);
RvSipAppTransmitterHandle

DESCRIPTION
The declaration of an application handle to a transmitter. This handle is used by the application to associate SIP Stack transmitter with an application transmitter object. The application gives the application handle when a new transmitter is created. The SIP Stack will give this handle back to the application in every callback function.

SYNTAX
RV_DECLARE_HANDLE(RvSipAppTransmitterHandle);
The Transmitter type definitions are:

- RvSipTransmitterState
- RvSipTransmitterReason
- RvSipTransmitterNewConnInUseStateInfo
- RvSipTransmitterRegExpFlag
- RvSipTransmitterRegExpMatch
- RvSipTransmitterRegExpResolutionParams
RvSipTransmitterState

DESCRIPTION

The Transmitter state machine represents the state of the transmitter in the message sending activity. The RvSipTransmitterStateChangedEv() callback reports transmitter state changes and state change reasons. The state change reason indicates the reason for the new state of the transmitter. The transmitter associates with the states listed below.

Note For a detailed description of the transmitter states, see the Working with Transmitters chapter in the HP-UX C SIP Stack Programmer’s Guide.

SYNTAX

typedef enum{
    RVSIP_TRANSMITTER_STATE_UNDEFINED = -1,
    RVSIP_TRANSMITTER_STATE_IDLE,
    RVSIP_TRANSMITTER_STATE_RESOLVING_ADDR,
    RVSIP_TRANSMITTER_STATE_FINAL_DEST_RESOLVED,
    RVSIP_TRANSMITTER_STATE_ON_HOLD,
    RVSIP_TRANSMITTER_STATE_NEW_CONN_IN_USE,
    RVSIP_TRANSMITTER_STATE_READY_FOR_SENDING,
    RVSIP_TRANSMITTER_STATE_MSG_SEND_FAILURE,
    RVSIP_TRANSMITTER_STATE_MSG_SENT,
    RVSIP_TRANSMITTER_STATE_TERMINATED
}RvSipTransmitterState;
Transmitter Type Definitions
RvSipTransmitterReason

RvSipTransmitterReason

DESCRIPTION
A transmitter has an inner state. The state changes according to the state machine defined for the transmitter. Each state change is explained by one of the reasons enumerated below.

SYNTAX
typedef enum{
    RVSIP_TRANSMITTER_REASON_UNDEFINED = -1,
    RVSIP_TRANSMITTER_REASON_NETWORK_ERROR,
    RVSIP_TRANSMITTER_REASON_CONNECTION_ERROR,
    RVSIP_TRANSMITTER_REASON_OUT_OF_RESOURCES,
    RVSIP_TRANSMITTER_REASON_NEW_CONN_CREATED,
    RVSIP_TRANSMITTER_REASON_CONN_FOUND_IN_HASH,
    RVSIP_TRANSMITTER_REASON_USER_COMMAND
}RvSipTransmitterReason;

PARAMETERS

RVSIP_TRANSMITTER_REASON_UNDEFINED
The reason is undefined.

RVSIP_TRANSMITTER_REASON_NETWORK_ERROR
A network error occurred while trying to send the request or during the DNS procedure.

RVSIP_TRANSMITTER_REASON_CONNECTION_ERROR
An error occurred on the connection that was used to send the message.

RVSIP_TRANSMITTER_REASON_OUT_OF_RESOURCES
The message cannot be sent because of a lack of resources.

RVSIP_TRANSMITTER_REASON_NEW_CONN_CREATED
Indicates that the transmitter created a new connection.
RVSIP_TRANSMITTER_REASON_CONN_FOUND_IN_HASH

Indicates that the transmitter found an existing connection in the connection hash and that the transmitter is about to use the connection.

RVSIP_TRANSMITTER_REASON_USER_COMMAND

The state was reached due to a user command.
RvSipTransmitterNewConnInUseStateInfo

DESCRIPTION
This structure is used in the state
RVSIP_TRANSMITTER_STATE_NEW_CONN_IN_USE to supply
additional information regarding the state.

SYNTAX
typedef struct{
    RvSipTransportConnectionHandle hConn;
}RvSipTransmitterNewConnInUseStateInfo;

PARAMETERS

hConn
The connection handle.
RvSipTransmitterRegExpFlag

**DESCRIPTION**

The regular expression parsing flags to be applied during the resolution of ENUM addresses, which involves regular expression phrases.

**SYNTAX**

```c
typedef enum{
    RVSIP_TRANSMITTER_REGEXP_FLAG_NONE = 0x00,
    RVSIP_TRANSMITTER_REGEXP_FLAG_NO_CASE = 0x01
}RvSipTransmitterRegExpFlag;
```

**PARAMETERS**

**RVSIP_TRANSMITTER_REGEXP_FLAG_NONE**

No regular expression parsing flags.

**RVSIP_TRANSMITTER_REGEXP_FLAG_NO_CASE**

Match regular expression phrases by ignoring their case.
RvSipTransmitterRegExpMatch

DESCRIPTION
This structure is used to notify the SIP Stack of the substring matches as an output of regular expression string parsing.

SYNTAX
typedef struct{
    RvInt32 startOffSet;
    RvInt32 endOffSet;
}RvSipTransmitterRegExpMatch;

PARAMETERS

startOffSet
If the value is not –1, indicates the start offset of the next largest substring match within a parsed string.

endOffSet
Indicates the end offset of the substring match, relative to the startOffSet value.
RvSipTransmitterRegExpResolutionParams

DESCRIPTION
This structure is used to exchange data between the SIP Stack and the application above the SIP Stack during an ENUM address resolution process. As known, this resolution process might require a stage of regular expression parsing, which takes place in the application layer. The SIP Stack uses most of the structure members as input parameters of the regular expression parsing process, and one of them is used as an output parameter of the process.

SYNTAX

typedef struct{
    RvChar *strRegExp;
    RvChar *strString;
    RvInt32 matchSize;
    RvSipTransmitterRegExpFlag eFlags;
    RvSipTransmitterRegExpMatch *pMatches;
}RvSipTransmitterRegExpResolutionParams;

PARAMETERS

strRegExp
A regular expression to be matched to the parsed address string. This is an input parameter of the regular expression parsing session of the application.

strString
The unresolved address string to be parsed, using a regular expression. This is an input parameter of the regular expression parsing session of the application.

matchSize
The size of the matching substrings array. This value inhibits the number of regular expression matching phrases, which are filled within the pMatch array by the application. This is an input parameter of the regular expression parsing session of the application.
Transmitter Type Definitions

**eFlags**
Regular expression parsing flags to be used when applying the parsed string. This is an input parameter of the regular expression parsing session of the application.

**pMatches**
The array of RvSipTransmitterRegExpMatch elements, which includes the parsing process output within substrings form. These substrings are the typical regular expression parsing output. Consequently, the first match consists of the whole phrase (taken from the parsed address string) that can match the whole regular expression, including its substrings. The following matches are substrings of the parsed string, which can be matched to the different regular expression substrings. Each array element represents one matching substring. If there are less matching substrings than the given matchSize, they are filled sequentially in the array and the rest of the free elements are marked with −1 offset values. For more information, see RvSipTransmitterRegExpMatch.
Transmitter Callback Functions

**TRANSMITTER CALLBACK FUNCTIONS**

The Transmitter callback functions are:

- `RvSipTransmitterStateChangedEv()`
- `RvSipTransmitterOtherURLAddressFoundEv()`
- `RvSipTransmitterEvHandlers`
- `RvSipTransmitterMgrRegExpResolutionNeededEv()`
- `RvSipTransmitterMgrEvHandlers()`
Transmitter Callback Functions

RvSipTransmitterStateChangedEv()

**DESCRIPTION**

Notifies the application of a transmitter state change. Each step of the message sending process of the transmitter is represented by a state. The application is notified of this state through the RvSipTransmitterStateChangedEv() callback. For each state change, the new state is supplied with the reason for the new state, and the message object when valid. An additional parameter, pExtraInfo, will hold specific state information. Most of the states are informative only. The final message sending states (MSG_SENT and MSG_SEND_FAILURE) indicate that the application should now terminate the transmitter.

**Note** Currently, the pExtraInfo parameter is used only in the NEW_CONN_IN_USE state.

**PARAMETERS**

- **hTrx**
  The SIP Stack transmitter handle.

- **hAppTrx**
  The application handle for this transmitter.

- **eState**
  The new transmitter state.

- **eReason**
  The reason for the state change.

- **hMsg**
  When the state relates to the outgoing message, the message is supplied.

- **pExtraInfo**
  Specific information for the new state.
Transmitter Callback Functions
RvSipTransmitterStateChangedEv()

**SYNTAX**

typedef void (RVCALLCONV * RvSipTransmitterStateChangedEv)(
    IN RvSipTransmitterHandle hTrx,
    IN RvSipAppTransmitterHandle hAppTrx,
    IN RvSipTransmitterState eState,
    IN RvSipTransmitterReason eReason,
    IN RvSipMsgHandle hMsg,
    IN void* pExtraInfo);

**RETURN VALUES**

None.
Transmitter Callback Functions

RvSipTransmitterOtherURLAddressFoundEv()

DESCRIPTION

Notifies the application that a message needs to be sent and the destination address is a URL type that is currently not supported by the SIP Stack (for example a TEL URL). The application must convert the URL to a SIP URL for the message to be sent.

SYNTAX

typedef RvStatus
(RVCALLCONV *RvSipTransmitterOtherURLAddressFoundEv)(
    IN RvSipTransmitterHandle hTrx,
    IN RvSipAppTransmitterHandle hAppTrx,
    IN RvSipMsgHandle hMsg,
    IN RvSipAddressHandle hAddress,
    OUT RvSipAddressHandle hSipURLAddress,
    OUT RvBool *pbAddressResolved);

PARAMETERS

hTrx

The SIP Stack transmitter handle.

hAppTrx

The application handle for this transmitter.

hMsg

The message that includes the other URL address.

hAddress

The handle to the unsupported address to be converted.

hSipURLAddress

The handle to the SIP URL address. This is an empty address object that the application should fill.
Transmitter Callback Functions
RvSipTransmitterOtherURLAddressFoundEv()

pbAddressResolved

Indicates whether or not the SIP URL address was filled.
RvSipTransmitterEvHandlers

DESCRIPTION
A structure with function pointers to the module callbacks. You supply this structure when creating a new transmitter.

SYNTAX
typedef struct{
    RvSipTransmitterStateChangedEv pfnStateChangedEv;
    RvSipTransmitterOtherURLAddressFoundEv pfnOtherURLAddressFoundEv;
}RvSipTransmitterEvHandlers;

PARAMETERS

pfnStateChangedEvHandler
Notifies of a transmitter state change event.

pfnOtherURLAddressFoundEvHandler
Notifies the application that another URL address was found and has to be converted to a SIP URL address.
**Transmitter Callback Functions**

**RvSipTransmitterMgrRegExpResolutionNeededEv()**

**DESCRIPTION**

When the SIP Stack 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**

```c
typedef RvStatus
   (RVCALLCONV *RvSipTransmitterMgrRegExpResolutionNeededEv)(
      IN RvSipTransmitterMgrHandle hTrxMgr,
      IN void* pAppTrxMgr,
      IN RvSipTransmitterHandle hTrx,
      IN RvSipAppTransmitterHandle hAppTrx,
      INOUT RvSipTransmitterRegExpResolutionParams* pRegExpParams);
```

**PARAMETERS**

**hTrxMgr**

The SIP Stack `TransmitterMgr` handle.

**pAppTrxMgr**

The application handle to the SIP Stack `TransmitterMgr`.

**hTrx**

A SIP Stack `transmitter` handle. This parameter may be NULL if the `transmitter` is not owned by the application.

**hAppTrx**

The application handle for this `transmitter`. This parameter may be NULL if the `transmitter` is not owned by the application.
Transmitter Callback Functions
RvSipTransmitterMgrRegExpResolutionNeededEv()

pRegExpParams
A structure that holds the information for the regular expression parsing process of the application. (For more information, see RvSipTransmitterRegExpResolutionParams.) The pMatches structure member of RvSipTransmitterRegExpResolutionParams should be filled in by this callback with substring match offsets. Any unused structure elements should contain the –1 offset values. Each startOffSet element that is not –1 indicates the start offset of the next largest substring match within the unresolved address string. The relative endOffSet element indicates the end offset of the match.
RvSipTransmitterMgrEvHandlers()

DESCRIPTION
A structure with function pointers to the module manager callbacks. You supply this structure to the TransmitterMgr using RvSipTransmitterMgrSetEvHandlers().

SYNTAX
typedef struct{
    RvSipTransmitterMgrRegExpResolutionNeededEv
    pfnRegExpResolutionNeededEvHandler;
}RvSipTransmitterMgrEvHandlers;

PARAMETERS
pfnRegExpResolutionNeededEvHandler
Asks the application for regular expression parsing of an unresolved ENUM address.
Transmitter Callback Functions

RvSipTransmitterMgrEvHandlers()
REGISTER-CLIENT API TYPE DEFINITIONS

This section includes the SIP Stack Register-Client type definitions and callback functions defined in the RvSipRegClientTypes.h header file.

This section includes:
- Handle Type Definitions
- Register-Client Type Definitions
- Register-Client Callback Functions
Handle Type Definitions

The Handle type definitions are:
- RvSipRegClientMgrHandle
- RvSipRegClientHandle
- RvSipAppRegClientHandle
RvSipRegClientMgrHandle

**DESCRIPTION**
Declaration of a handle to a Register-clientMgr. The Register-clientMgr manages all register-clients and the handle is required by all manager object-related API functions, such as the creation of new register-client.

**SYNTAX**
RV_DECLARE_HANDLE(RvSipRegClientMgrHandle);

**PARAMETERS**
None.

**RETURN VALUES**
None.
**RvSipRegClientHandle**

**DESCRIPTION**
Declaration of a *register-client* handle. A *register-client* handle is needed by all Register-Client API functions and is used to reference the *register-client*.

**SYNTAX**
RV_DECLARE_HANDLE(RvSipRegClientHandle);

**PARAMETERS**
None.

**RETURN VALUES**
None.
**RvSipAppRegClientHandle**

**DESCRIPTION**

Declaration of application handle to a *register-client*. This handle is used by the application in order to associate SIP Stack *register-clients* with application *register-clients*. The application gives the application handle when a new *register-client* is created. The SIP Stack will give this handle back to the application in every callback function.

**SYNTAX**

```c
RV_DECLARE_HANDLE(RvSipAppRegClientHandle);
```

**PARAMETERS**

None.

**RETURN VALUES**

None.
Register-Client Type Definitions

The Register-Client type definitions are:

- RvSipRegClientState
- RvSipRegClientStateChangeReason
- RvSipRegClientEvHandlers
**RvSipRegClientState**

**DESCRIPTION**

Represents the state of the registration process. The state changes are reported using the `RvSipRegClientStateChangedEv()` callback function together with a state change reason that may indicate to the application how the new state was reached.

**Note** For a detailed description of the `register-client` states, see the Working with Register-Clients chapter in the HP-UX C SIP Stack Programmer's Guide.

**SYNTAX**

```c
typedef enum{
    RVSIP_REG_CLIENT_STATE_UNDEFINED = -1,
    RVSIP_REG_CLIENT_STATE_IDLE,
    RVSIP_REG_CLIENT_STATE_TERMINATED,
    RVSIP_REG_CLIENT_STATE_REGISTERING,
    RVSIP_REG_CLIENT_STATE_REDIRECTED,
    RVSIP_REG_CLIENT_STATE_UNAUTHENTICATED,
    RVSIP_REG_CLIENT_STATE_REGISTERED,
    RVSIP_REG_CLIENT_STATE_FAILED,
    RVSIP_REG_CLIENT_STATE_MSG_SEND_FAILURE
}RvSipRegClientState;
```
RvSipRegClientStateChangeReason

DESCRIPTION

A register-client has an inner state. The state changes according to the state machine defined for the register-client. Each state change is explained by one of the reasons enumerated below.

SYNTAX

typedef enum {
    RVSIP_REG_CLIENT_REASON_UNDEFINED = -1,
    RVSIP_REG_CLIENT_REASON_USER_REQUEST,
    RVSIP_REG_CLIENT_REASON_RESPONSE_SUCCESSFUL_RECVD,
    RVSIP_REG_CLIENT_REASON_RESPONSE_REDIRIRECTION_RECVD,
    RVSIP_REG_CLIENT_REASON_RESPONSE_UNAUTHENTICATED_RECVD,
    RVSIP_REG_CLIENT_REASON_RESPONSE_REQUEST_FAILURE_RECVD,
    RVSIP_REG_CLIENT_REASON_RESPONSE_SERVER_FAILURE_RECVD,
    RVSIP_REG_CLIENT_REASON_RESPONSE_GLOBAL_FAILURE_RECVD,
    RVSIP_REG_CLIENT_REASON_LOCAL_FAILURE,
    RVSIP_REG_CLIENT_REASON_TRANSACTION_TIMEOUT,
    RVSIP_REG_CLIENT_REASON_NORMAL_TERMINATION,
    RVSIP_REG_CLIENT_REASON_GIVE_UP_DNS,
    RVSIP_REG_CLIENT_REASON_NETWORK_ERROR,
    RVSIP_REG_CLIENT_REASON_503_RECEIVED,
    RVSIP_REG_CLIENT_REASON_CONTINUE_DNS
} RvSipRegClientStateChangeReason;

PARAMETERS

RVSIP_REG_CLIENT_REASON_UNDEFINED

The reason is undefined.

RVSIP_REG_CLIENT_REASON_USER_REQUEST

The change in state was in reaction to a request of the user.
**RVSIP_REG_CLIENT_REASON_RESPONSE_SUCCESSFUL_RECEIVED**

A 2xx was received.

**RVSIP_REG_CLIENT_REASON_RESPONSE_REDIRECT_RECEIVED**

A 3XX was received.

**RVSIP_REG_CLIENT_REASON_RESPONSE_UNAUTHENTICATED_RECEIVED**

A 401 or 407 was received.

**RVSIP_REG_CLIENT_REASON_RESPONSE_REQUEST_FAILURE_RECVD**

- A 401 or 407 with an invalid Authentication challenge was received.
- A 4xx other than a 401 or 407 was received.
- A 401 or 407 is not supported.

**RVSIP_REG_CLIENT_REASON_RESPONSE_SERVER_FAILURE_RECVD**

A 5xx response was received.

**RVSIP_REG_CLIENT_REASON_RESPONSE_GLOBAL_FAILURE_RECVD**

A 6xx response was received.

**RVSIP_REG_CLIENT_REASON_LOCAL_FAILURE**

- Failures as a response in an un-expcted state.
- Failed to check the validity of the authentication object.
- The status code of the final response is not legal.
- In the case of RVSIP_TRANSC_REASON_OUT_OF_RESOURCES, the register-client transaction is terminated.

**RVSIP_REG_CLIENT_REASON_TRANSACTION_TIMEOUT**

The register transaction had a time-out. This means that no final response was received.

**RVSIP_REG_CLIENT_REASON_NORMAL_TERMINATION**

The register-client terminated normally.
**Register-Client Type Definitions**

RvSipRegClientStateChangeReason

- **RVSIP_REG_CLIENT_REASON_GIVE_UP_DNS**
  The local party decided to give up sending a message after its previous send attempt failed (DNS feature).

- **RVSIP_REG_CLIENT_REASON_NETWORK_ERROR**
  One of the reasons for moving to the MSG_SEND_FAILURE state. The message failed to be sent due to a network error.

- **RVSIP_REG_CLIENT_REASON_503_RECEIVED**
  One of the reasons for moving to the MSG_SEND_FAILURE state. The message received a 503 response.

- **RVSIP_REG_CLIENT_REASON_CONTINUE_DNS**
  The local party decided to send a message to the next address in its transaction DNS list, after its previous send attempt failed (DNS feature).
RvSipRegClientEvHandlers

DESCRIPTION
Structure with function pointers to the Register-Client module callbacks. This structure is used to set the application callbacks in the RvSipRegClientMgrSetEvHandlers() function.

SYNTAX

typedef struct{
    RvSipRegClientStateChangedEv pfnStateChangedEvHandler;
    RvSipRegClientMsgToSendEv pfnMsgToSendEvHandler;
    RvSipRegClientMsgReceivedEv pfnMsgReceivedEvHandler;
    RvSipRegClientOtherURLAddressFoundEv pfnOtherURLAddressFoundEvHandler;
    RvSipRegClientFinalDestResolvedEv pfnFinalDestResolvedEvHandler;
    RvSipRegClientNewConnInUseEv pfnNewConnInUseEvHandler;
} RvSipRegClientEvHandlers;

PARAMETERS

pfnStateChangedEvHandler
Notification of a register-client state change.

pfnMsgToSendEvHandler
Notification that a message is about to be sent.

pfnMsgReceivedEvHandler
Notification that a message was received.
Register-Client Type Definitions
RvSipRegClientEvHandlers

**pfnOtherURLAddressFoundEvHandler**
Notifies the application that the destination address is not a SIP URL. The application must convert the address to a SIP URL address for the message to be sent.

**pfnFinalDestResolvedEvHandler**
Notifies that the *register-client* 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. The application can use the Transaction API function to get the destination address.

**pfnNewConnInUseEvHandler**
Notifies that the *register-client* is now using a new *connection*. The *connection* can be a totally new *connection*, or a suitable *connection* that was found in the hash.
The Register-Client callback functions are:

- `RvSipRegClientStateChangedEv()`  
- `RvSipRegClientMsgToSendEv()`  
- `RvSipRegClientMsgReceivedEv()`  
- `RvSipRegClientOtherURLAddressFoundEv()`  
- `RvSipRegClientFinalDestResolvedEv()`  
- `RvSipRegClientNewConnInUseEv()`
**RvSipRegClientStateChangedEv()**

**DESCRIPTION**

Notifies the application of a *register-client* state change. For each state change the new state is supplied and the reason for the state change is also given.

**SYNTAX**

```c
typedef void (RVCALLCONV * RvSipRegClientStateChangedEv)(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipAppRegClientHandle hAppRegClient,
    IN RvSipRegClientState eNewState,
    IN RvSipRegClientStateChangeReason eReason);
```

**PARAMETERS**

- **hRegClient**
  The SIP Stack *register-client* handle.

- **hAppRegClient**
  The application handle for this *register-client*.

- **eNewState**
  The new state of the *register-client*.

- **eReason**
  The reason for the change in state.

**RETURN VALUES**

None.
RvSipRegClientMsgToSendEv()

DESCRIPTION
An event indicating that a new register-client related outgoing message is about to be sent.

SYNTAX
typedef RvStatus (RVCALLCONV * RvSipRegClientMsgToSendEv)(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipAppRegClientHandle hAppRegClient,
    IN RvSipMsgHandle hMsg);

PARAMETERS
hRegClient
The SIP Stack register-client handle.

hAppRegClient
The application handle for this register-client.

hMsg
The handle to the outgoing message.

RETURN VALUES
If you return a value other than RV_OK, the message will not be sent.
Register-Client Callback Functions
RvSipRegClientMsgReceivedEv()

RvSipRegClientMsgReceivedEv()

DESCRIPTION
An event indicating that a new register-client related incoming message was received.

SYNTAX
typedef RvStatus (RVCALLCONV * RvSipRegClientMsgReceivedEv)(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipAppRegClientHandle hAppRegClient,
    IN RvSipMsgHandle hMsg);

PARAMETERS

hRegClient
The SIP Stack register-client handle.

hAppRegClient
The application handle for this register-client.

hMsg
The handle to the incoming message.

RETURN VALUES
If you return a value other than RV_OK, the message will be ignored and the underlying transaction will terminate.
Register-Client Callback Functions
RvSipRegClientOtherURLAddressFoundEv()

**RvSipRegClientOtherURLAddressFoundEv()**

**DESCRIPTION**

Notifies the application that a message needs to be sent and the destination address is a URL type that is currently not supported by the SIP Stack. The URL has to be converted to a SIP URL for the message to be sent.

**SYNTAX**

typedef RvStatus
(RVCALLCONV *RvSipRegClientOtherURLAddressFoundEv)(
  N RvSipRegClientHandle hRegClient,
  IN RvSipAppRegClientHandle hAppRegClient,
  IN RvSipMsgHandle hMsg,
  IN RvSipAddressHandle hAddress,
  OUT RvSipAddressHandle hSipURLAddress,
  OUT RvBool* bAddressResolved);

**PARAMETERS**

**hRegClient**

The SIP Stack register-client handle.

**hAppRegClient**

The application handle for this register-client.

**hMsg**

The message that includes the other URL address.

**hAddress**

The handle to unsupported address to be converted.

**hSipURLAddress**

The handle to the SIP URL address. This is an empty address object that the application should fill.
Register-Client Callback Functions
RvSipRegClientOtherURLAddressFoundEv()

bAddressResolved

RV_TRUE if the SIP URL address was filled. Otherwise, RV_FALSE.
**Register-Client Callback Functions**

**RvSipRegClientFinalDestResolvedEv()**

**DESCRIPTION**

Notifies the application that the *register-client* 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. When this callback is called, the application can query the *transaction* about the destination address using the `RvSipTransactionGetCurrentDestAddress()` function. If the application wishes, it can update the sent-by part of the top-most Via header. The application must not update the branch parameter.

**SYNTAX**

```c
typedef RvStatus (RVCALLCONV *
    RvSipRegClientFinalDestResolvedEv)(
    IN RvSipRegClientHandle hRegClient,
    IN RvSipAppRegClientHandle hAppRegClient,
    IN RvSipTranscHandle hTransc,
    IN RvSipMsgHandle hMsgToSend);
```

**PARAMETERS**

- **hRegClient**
  The handle to the *register-client*.

- **hAppRegClient**
  The application handle for this *register-client*.

- **hTransc**
  The *transaction* handle.

- **hMsgToSend**
  The handle to the outgoing message.

**RETURN VALUES**

Returns `RvStatus`. 
If the application returns a value other than RV_OK, the message will not be sent. The transaction will terminate with an error and the register-client will move to the FAILED state.
RvSipRegClientNewConnInUseEv()

DESCRIPTION
Notifies the application that the transaction is now using a new connection. The connection can be a totally new one or a suitable one that was found in the hash.

SYNTAX

typedef RvStatus
    (RVCALLCONV *RvSipRegClientNewConnInUseEv) (
    IN RvSipRegClientHandle hRegClient,
    IN RvSipAppRegClientHandle hAppRegClient,
    IN RvSipTransportConnectionHandle hConn,
    IN RvBool bNewConnCreated);

PARAMETERS

hRegClient
The handle to the register-client.

hAppRegClient
The application handle for this register-client.

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 RV_OK. (The returned status is ignored in the current SIP Stack version.)
Register-Client Callback Functions
RvSipRegClientNewConnInUseEv()
This section includes the SIP Stack Authenticator type definitions and callback functions defined in the RvSipAuthenticator.h header file.

This section includes:
- Handle Type Definitions
- Authenticator Type Definitions
- Authenticator Callback Functions
HANDLE TYPE DEFINITIONS

The handle type definition is:

- RvSipAuthenticatorHandle
- RvSipAppAuthenticatorHandle
- RvSipAuthObjHandle
RvSipAuthenticatorHandle

DESCRIPTION
Declares the handle to the AuthenticatorMgr.

SYNTAX
RV_DECLARE_HANDLE(RvSipAuthenticatorHandle);

PARAMETERS
None.

RETURN VALUES
None.
RvSipAppAuthenticatorHandle

DESCRIPTION
Declares the application handle to the authenticator.

SYNTAX
RV_DECLARE_HANDLE(RvSipAppAuthenticatorHandle);

PARAMETERS
None.

RETURN VALUES
None.
RvSipAuthObjHandle

**DESCRIPTION**

Declares the handle to an authentication-object. (An authentication-object represents a single Authentication header kept in a SIP Stack object.)

**SYNTAX**

```c
RV_DECLARE_HANDLE(RvSipAuthObjHandle);
```
The Authenticator type definition is:

- RvSipAuthenticatorEvHandlers
RvSipAuthenticatorEvHandlers

DESCRIPTION
A structure with function pointers to the hash algorithm callbacks. This structure is used to set the application callbacks in the RvSipAuthenticatorSetEvHandlers() function.

SYNTAX
typedef struct {
    RvSipAuthenticatorGetSharedSecretEv
        pfnGetSharedSecretAuthenticationHandler;
    RvSipAuthenticatorMD5ExEv
        pfnMD5AuthenticationExHandler;
    RvSipAuthenticatorMD5EntityBodyEv
        pfnMD5EntityBodyAuthenticationHandler;
    RvSipAuthenticatorUnsupportedChallengeEv
        pfnUnsupportedChallengeAuthenticationHandler;
    RvSipAuthenticatorNonceCountUsageEv
        pfnNonceCountUsageAuthenticationHandler;
    RvSipAuthenticatorAuthorizationReadyEv
        pfnAuthorizationReadyAuthenticationHandler;
} RvSipAuthenticatorEvHandlers;

PARAMETERS

    pfnGetSharedSecretAuthenticationHandler
    Notifies the application that there is a need for a username and password.

    pfnMD5AuthenticationExHandler
    Notifies the application that the MD5 algorithm should be used.

    pfnMD5EntityBodyAuthenticationHandler
    Asks the application for the H (entity-body) string, necessary for the A2 token calculation in the case of “auth-int” Quality-of-Protection authentication.
Authenticator Type Definitions

pfnUnsupportedChallengeAuthenticationHandler

Notifies the application about a challenge whose credentials cannot be prepared by the SIP Stack. The application can calculate the credentials itself, build the Authorization header in the message, using RvSipAuthorizationHeaderConstructInMsg(), and set the calculated credential into it.

pfnNonceCountUsageAuthenticationHandler

Notifies the application about the nonce count that will be used while preparing credentials. The application can change the nonce count value.

pfnAuthorizationReadyAuthenticationHandler

Notifies the application that an Authorization header was built, and is now ready for sending. The application can set additional information to the header in this callback.
The Authenticator callback functions are:

- RvSipAuthenticatorMD5ExEv()
- RvSipAuthenticatorGetSharedSecretEv()
- RvSipAuthenticatorMD5EntityBodyEv()
- RvSipAuthenticatorAuthorizationReadyEv()
- RvSipAuthenticatorUnsupportedChallengeEv()
- RvSipAuthenticatorNonceCountUsageEv()
RvSipAuthenticatorMD5ExEv()

DESCRIPTION
Notifies the application of the need to use the MD5 algorithm. The SIP Stack supplies the application with the input string for the algorithm and the application returns the output string of the MD5 algorithm.

SYNTAX
typedef void (RVCALLCONV * RvSipAuthenticatorMD5ExEv)(
    IN RvSipAuthenticatorHandle hAuthenticator,
    IN RvSipAppAuthenticatorHandle hAppAuthenticator,
    IN RPOOL_Ptr *pRpoolMD5Input,
    IN RvUint32 length,
    OUT RPOOL_Ptr *pRpoolMd5Output);

PARAMETERS

hAuthenticator
The handle to the authenticator.

hAppAuthenticator
The handle to the application authenticator handle.

pRpoolMD5Input
An RPOOL pointer to the MD5 input.

length
The length of the string inside the page.

pRpoolMd5Output
The output of the hash algorithm.

RETURN VALUES
None.
Authenticator Callback Functions

typedef void (RVCALLCONV * RvSipAuthenticatorGetSharedSecretEv)(
    IN RvSipAuthenticateHandle hAuthenticator,
    IN RvSipAppAuthenticatorHandle hAppAuthenticator,
    IN void* hObject,
    IN void* peObjectType,
    INOUT RPOOL_Ptr *pRpoolRealm,
    INOUT RPOOL_Ptr *pRpoolUserName,
    OUT RPOOL_Ptr *pRpoolPassword);

PARAMETERS

hAuthenticator
The handle to the authenticator.

hAppAuthenticator
The handle to the application authenticator.

hObject
The handle to the object, that is served by the authenticator (such as call-leg or register-client).
Authenticator Callback Functions
RvSipAuthenticatorGetSharedSecretEv()

**peObjectType**
The pointer to the variable that stores the type of the *hObject*. Use the following code to get the type:

```c
RvSipCommonStackObjectType eObjType = *peObjectType;
```

**pRpoolRealm**
The realm string in RPool_ptr format.

**pRpoolUserName**
The username string in RPool_ptr format. (This is an OUT parameter for the client, and an IN parameter for the server.)

**pRpoolPassword**
The password string in RPool_ptr format.

**RETURN VALUES**

None.
Authenticator Callback Functions
RvSipAuthenticatorMD5EntityBodyEv()

RvSipAuthenticatorMD5EntityBodyEv()

DESCRIPTION

Notifies the application that it should supply the hash result made on the message body MD5 (entity-body). The message body hash value is needed when the required Quality-of-Protection (Qop) is “auth-int”.

Note  This callback supplies the message object as a parameter. However, it is called before the msgToSent callback of SIP Stack objects. If your code adds the message body in the msgToSent callback, the body will not be available when this callback is called. If you wish the message object to include the body at this stage, you must use the outbound message mechanism to add the body.

SYNTAX

typedef void(RVCALLCONV *
RvSipAuthenticatorMD5EntityBodyEv)(
    IN RvSipAuthenticatorHandle hAuthenticator,
    IN RvSipAppAuthenticatorHandle hAppAuthenticator,
    IN void* hObject,
    IN void* peObjectType,
    IN RvSipMsgHandle hMsg,
    OUT RPOOL_Ptr *pRpoolMD5Output;

PARAMETERS

hAuthenticator
The handle to the authenticator.

hAppAuthenticator
The handle to the application authenticator.

hObject
The handle to the object that is served by the authenticator (such as call-leg or register-client).
Authenticator Callback Functions
RvSipAuthenticatorMD5EntityBodyEv()

**peObjectType**
A pointer to the variable that stores the type of the *hObject*. Use the following code to get the type:

```c
RvSipCommonStackObjectType eObjType = *peObjectType;
```

**hMsg**
The handle to the message that it is now being sent and that will include the user credentials.

**pRpoolMD5Output**
The MD5 of the message body in RPOOL_Ptr format.

**RETURN VALUES**
None.
RvSipAuthenticatorAuthorizationReadyEv()

DESCRIPTION

Notifies the application that an Authorization header was fully built in the message. In this callback, the application may add additional information to the given Authorization header.

SYNTAX

typedef void(RVCALLCONV * 
RvSipAuthenticatorAuthorizationReadyEv)(
    IN RvSipAuthenticatorHandle hAuthenticator,
    IN RvSipAppAuthenticatorHandle hAppAuthenticator,
    IN void* hObject,
    IN void* peObjectType,
    IN RvSipAuthObjHandle hAuthObj,
    IN void* pAuthObjContext,
    IN RvSipAuthorizationHeaderHandle hAuthorizationHeader);

PARAMETERS

hAuthenticator

The handle to the authenticator.

hAppAuthenticator

The application handle stored in the authenticator.

hObject

The handle to the object, that is served by the authenticator (such as call-leg, register-client, and so on).

peObjectType

A pointer to the variable that stores the type of the hObject. Use following code to get the type:

RvSipCommonStackObjectObjectType eObjType = *peObjectType;
Authenticator Callback Functions

RvSipAuthenticatorAuthorizationReadyEv()

RvSipAuthObjHandle
The handle to the authentication-object, related to this new Authorization header.

pAuthObjContext
The context of the application of the related authentication-object.

hAuthorizationHeader
The Authorization header which is already filled.

RETURN VALUES
None.
**AUTHENTICATOR API TYPE DEFINITIONS**

**RvSipAuthenticatorUnsupportedChallengeEv()**

**DESCRIPTION**

Notifies the application about a challenge whose credentials cannot be prepared by the SIP Stack. The application can calculate the credentials itself, build the Authorization Header in the message, using RvSipAuthorizationHeaderConstructInMsg(), and set the calculated credential into it. For more information on the RvSipAuthorizationHeaderConstructInMsg() function, see the Authorization Header Functions chapter in the HP-UX C SIP Stack Message Layer Reference Guide.

**SYNTAX**

```c
typedef void (RVCALLCONV *
RvSipAuthenticatorUnsupportedChallengeEv)(
    IN RvSipAuthenticatorHandle hAuthenticator,
    IN RvSipAppAuthenticatorHandle hAppAuthenticator,
    IN void* hObject,
    IN void* peObjectType,
    IN RvSipAuthenticationHeaderHandle hAuthenticationHeader,
    IN RvSipMsgHandle hMsg);
```

**PARAMETERS**

- **hAuthenticator**
  The handle to the authenticator.

- **hAppAuthenticator**
  The handle to the application authenticator.

- **hObject**
  The handle to the object that is served by the authenticator (such as call-leg or register-client).
Authenticator Callback Functions
RvSipAuthenticatorUnsupportedChallengeEv()

**peObjectType**
The pointer to the variable that stores the type of the *hObject*. Use the following code to get the type:

```c
RvSipCommonStackObjectType eObjType = *peObject;  
```

**hAuthenticationHeader**
The handle to the Authentication header containing the unsupported challenge.

**hMsg**
The handle to the message where the application should set the credentials, if it wishes to do so.

**RETURN VALUES**
None.
RvSipAuthenticatorNonceCountUsageEv()

DESCRIPTION

Notifies the application about the value of the nonceCount parameter that the SIP Stack is going to use when calculating credentials. The application can change this value to fit more precise management of the nonceCount. The SIP Stack does not check the uniqueness of the used NONCE in the realm or by the different object.

SYNTAX

typedef void (RVCALLCONV *
RvSipAuthenticatorNonceCountUsageEv)(
    IN  RvSipAuthenticatorHandle hAuthenticator,
    IN  RvSipAppAuthenticatorHandle hAppAuthenticator,
    IN  void* hObject,
    IN  void* peObjectType,
    IN  RvSipAuthenticationHeaderHandle hAuthenticationHeader,
    INOUT RvInt32* pNonceCount);

PARAMETERS

hAuthenticator
The handle to the authenticator.

hAppAuthenticator
The handle to the application authenticator.

hObject
The handle to the object that is served by the authenticator (such as call-leg or register-client).

peObjectType
The pointer to the variable that stores the type of the hObject. Use the following code to get the type:
RvSipCommonStackObjectType eObjType = *peObjectType;
Authenticator Callback Functions
RvSipAuthenticatorNonceCountUsageEv()

**hAuthenticationHeader**

The handle to the Authentication header, containing the unsupported challenge.

**pNonceCount**

A pointer to the nonceCount value set by the application to be used by the SIP Stack for calculation of the credentials.

**RETURN VALUES**

None.
EVENT NOTIFICATION API TYPE DEFINITIONS

This section includes the SIP Stack Event Notification type definitions and callback functions type definitions defined in the `RvSipSubscriptionTypes.h` header file.

This section includes:

- Handle Type Definitions
- Subscription Type Definitions
- Subscription Callback Functions
- Notification Callback Functions
- Authentication Callback Functions
- Message Callback Functions
- REFER Callback Functions
- General Callback Functions
Handle Type Definitions

**HANDLE TYPE DEFINITIONS**

The handle type definitions are:

- RvSipSubsMgrHandle
- RvSipSubsHandle
- RvSipAppSubsHandle
- RvSipNotifyHandle
- RvSipAppNotifyHandle
RvSipSubsMgrHandle

DESCRIPTION
The declaration of handle to a SubscriptionMgr. The SubscriptionMgr manages all subscriptions and its handle is needed in all manager-related functions, such as the creation of new subscriptions.

SYNTAX
DECLARE_VOID_HANDLE(RvSipSubsMgrHandle);

PARAMETERS
None.

RETURN VALUES
None.
RvSipSubsHandle

DESCRIPTION
The handle to a subscription. A subscription handle is needed in all Subscription API functions and is used to reference the correct subscription.

SYNTAX
RV_DECLARE_HANDLE(RvSipSubsHandle);

PARAMETERS
None.

RETURN VALUES
None.
RvSipAppSubsHandle

DESCRIPTION

The application handle to a subscription. The application uses this handle to associate SIP Stack subscriptions with application subscriptions. The application gives the application handle when a new subscription is created. The SIP Stack gives this handle back to the application in every callback function.

SYNTAX

RV_DECLARE_HANDLE(RvSipAppSubsHandle);

PARAMETERS

None.

RETURN VALUES

None.
RvSipNotifyHandle

DESCRIPTION
The handle to a notify object. A notify handle is needed in all Notification API functions and is used to reference the correct notification.

SYNTAX
RV_DECLARE_HANDLE(RvSipNotifyHandle);

PARAMETERS
None.

RETURN VALUES
None.
**RvSipAppNotifyHandle**

**DESCRIPTION**

The application handle to a notify object. The application uses this handle to associate SIP Stack *notifications* with application *notification*. The application gives the application handle when a new *notification* is created. The SIP Stack gives this handle back to the application in every callback function.

**SYNTAX**


```c
RV_DECLARE_HANDLE(RvSipAppNotifyHandle);
```

**PARAMETERS**

None.

**RETURN VALUES**

None.
Subscription Type Definitions

The Subscription type definitions are:

- RvSipSubsState
- RvSipSubsStateChangeReason
- RvSipSubscriptionType
- RvSipSubsNotifyStatus
- RvSipSubsNotifyReason
- RvSipSubsEventPackageType
- RvSipSubsReferNotifyReadyReason
- RvSipSubsEvHandlers
**RvSipSubsState**

**DESCRIPTION**

Represents a state in a subscription. The state changes are reported using the RvSipSubsStateChangedEv() callback function together with a state change reason that may indicate to the application how the new state was reached. The subscription states are divided into the following three groups:

- Subscriber states
- Notifier states
- Common states

**Note** a detailed description of the subscription states, see the Event Notification chapter in the HP-UX C SIP Stack Programmer’s Guide.

**SYNTAX**

typedef enum{
  RVSIP_SUBS_STATE_UNDEFINED = -1,
  RVSIP_SUBS_STATE_IDLE,
  RVSIP_SUBS_STATE_SUBS_SENT,
  RVSIP_SUBS_STATE_REDIRECTED,
  RVSIP_SUBS_STATE_UNAUTHENTICATED,
  RVSIP_SUBS_STATE_2XX_RCVD,
  RVSIP_SUBS_STATE_NOTIFY_BEFORE_2XX_RCVD,
  RVSIP_SUBS_STATE_REFRESHING,
  RVSIP_SUBS_STATE_REFRESH_RCVD,
  RVSIP_SUBS_STATE_UNSUBSCRIBING,
  RVSIP_SUBS_STATE_UNSUBSCRIBE_RCVD,
  RVSIP_SUBS_STATE_2XX_RCVD,
  RVSIP_SUBS_STATE_SUBS_RCVD,
  RVSIP_SUBS_STATE_ACTIVATE,
  RVSIP_SUBS_STATE_TERMINATING,
  RVSIP_SUBS_STATE_PENDING,
  RVSIP_SUBS_STATE_ACTIVE,
  RVSIP_SUBS_STATE_MSG_SEND_FAILURE,
}
Subscription Type Definitions
RvSipSubsState

RVSIP_SUBS_STATE_TERMINATED

}RvSipSubsState;
**RvSipSubsStateChangeReason**

**DESCRIPTION**

A *subscription* has an inner state. The state changes according to the state machine defined for the *subscription*. Each state change is explained by one of the reasons enumerated below.

**SYNTAX**

typedef enum {
    RVSIP_SUBS_REASON_UNDEFINED = -1,
    RVSIP_SUBS_REASON_LOCAL_REJECT,
    RVSIP_SUBS_REASON_LOCAL_FAILURE,
    RVSIP_SUBS_REASON_REMOTE_REJECT,
    RVSIP_SUBS_REASON_481_RCVD,
    RVSIP_SUBS_REASON_UNAUTHENTICATED,
    RVSIP_SUBS_REASON_UNSUBSCRIBE_UNAUTHENTICATED,
    RVSIP_SUBS_REASON_UNSUBSCRIBE_REJECTED,
    RVSIP_SUBS_REASON_NO_NOTIFY_TIME_OUT,
    RVSIP_SUBS_REASON.TRANSC_TIME_OUT,
    RVSIP_SUBS_REASON.TRANSC_ERROR,
    RVSIP_SUBS_REASON_NO_TIMERS,
    RVSIP_SUBS_REASON.NETWORK_ERROR,
    RVSIP_SUBS_REASON.503_RECEIVED,
    RVSIP_SUBS_REASON.GIVE_UP_DNS,
    RVSIP_SUBS_REASON.CONTINUE_DNS,
    RVSIP_SUBS_REASON.NOTIFY_ACTIVE_SENT,
    RVSIP_SUBS_REASON.NOTIFY_TERMINATED_SENT,
    RVSIP_SUBS_REASON.NOTIFY_TERMINATED.DEACTIVATED_RCVD,
    RVSIP_SUBS_REASON.NOTIFY_TERMINATED.PROBATION_RCVD,
    RVSIP_SUBS_REASON.NOTIFY_TERMINATED.REJECTED_RCVD,
    RVSIP_SUBS_REASON.NOTIFY_TERMINATED.TIMEOUT_RCVD,
    RVSIP_SUBS_REASON.NOTIFY_TERMINATED.GIVEUP_RCVD,
    RVSIP_SUBS_REASON.NOTIFY_TERMINATED.NORESOURCE_RCVD,
    RVSIP_SUBS_REASON.NOTIFY_2XX_SENT,
    RVSIP_SUBS_REASON.NOTIFY_2XX_RCVD,
}
Subscription Type Definitions
RsSipSubsStateChangeReason

RVSIP_SUBS_REASON_SUBS_TERMINATED,
RVSIP_SUBS_REASON_DIALOG_TERMINATION,
RVSIP_SUBS_REASONREFER_RCVD,
RVSIP_SUBS_REASONREFER_RCVD_WITH_REPLACES,
RVSIP_SUBS_REASON_ILLEGALREFER_MSG

}RsSipSubsStateChangeReason;

PARAMETERS

RVSIP_SUBS_REASON_UNDEFINED
The reason is undefined. This reason is used when there is no need for additional
information. For example, the SUBS_RCVD state with the UNDEFINED
reason means that the initial SUBSCRIBE request was received.

RVSIP_SUBS_REASON_LOCAL_REJECT
The local party rejected the subscription.

RVSIP_SUBS_REASON_LOCAL_FAILURE
The local party failed. For example, it failed to encode the message while
sending a request.

RVSIP_SUBS_REASON_REMOTE_REJECT
The remote party rejected the request with a 4xx-6xx response with the
exception of 401, 407 or 481.

RVSIP_SUBS_REASON_481_RCVD
A 481 response was received.

RVSIP_SUBS_REASON_UNAUTHENTICATED
A 401/407 response was received (for an initial SUBSCRIBE request or for a
REFRESH subscribe request).

RVSIP_SUBS_REASON_UNSUBSCRIBE_UNAUTHENTICATED
A 401/407 response was received on an UNSUBSCRIBE request.
RVSIP_SUBS_REASON_UNSUBSCRIBE_REJECTED
A reject response was received on an UNSUBSCRIBE request.

RVSIP_SUBS_REASON_NO_NOTIFY_TIME_OUT
The NoNotify timer expired. (This means that a NOTIFY request was not accepted. This timer is set after receiving a 2xx on an initial SUBSCRIBE or UNSUBSCRIBE request).

RVSIP_SUBS_REASON_TRANSC_TIME_OUT
A transaction timeout occurred.

RVSIP_SUBS_REASON_TRANSC_ERROR
A transaction error occurred.

RVSIP_SUBS_REASON_NO_TIMERS
There are no more timers to set.

RVSIP_SUBS_REASON_NETWORK_ERROR
A network error occurred.

RVSIP_SUBS_REASON_503_RECEIVED
A 503 response was received.

RVSIP_SUBS_REASON_GIVE_UP_DNS
The application called the RvSipSubsDNSGiveUp() function.

RVSIP_SUBS_REASON_CONTINUE_DNS
The application called the RvSipSubsDNSContinue() function.

RVSIP_SUBS_REASON_NOTIFY_ACTIVE_SENT
A NOTIFY request with “active” as the Subscription-State header value was sent.
Subscription Type Definitions
RvSipSubsStateChangeReason

**RVSIP_SUBS_REASON_NOTIFY_TERMINATED_SENT**
A NOTIFY request with a “terminated” Subscription-State header value was sent.

**RVSIP_SUBS_REASON_NOTIFY_TERMINATED_DEACTIVATED_RCVD**
A NOTIFY request was received with a “terminated” Subscription-State header value and “deactivated” reason.

**RVSIP_SUBS_REASON_NOTIFY_TERMINATED_PROBATION_RCVD**
A NOTIFY request was received with a “terminated” Subscription-State header value and “probation” reason.

**RVSIP_SUBS_REASON_NOTIFY_TERMINATED_REJECTED_RCVD**
A NOTIFY request was received with a “terminated” Subscription-State header value and “rejected” reason.

**RVSIP_SUBS_REASON_NOTIFY_TERMINATED_TIMEOUT_RCVD**
A NOTIFY request was received with a “terminated” Subscription-State header value and “timeout” reason.

**RVSIP_SUBS_REASON_NOTIFY_TERMINATED_GIVEUP_RCVD**
A NOTIFY request was received with a “terminated” Subscription-State header value and “giveup” reason.

**RVSIP_SUBS_REASON_NOTIFY_TERMINATED_NORESOURCE_RCVD**
A NOTIFY request was received with a “terminated” Subscription-State header value and “no resource” reason.

**RVSIP_SUBS_REASON_NOTIFY_2XX_SENT**
A 2xx response to the NOTIFY request was sent.

**RVSIP_SUBS_REASON_NOTIFY_2XX_RCVD**
A 2xx for the NOTIFY request was received.

**RVSIP_SUBS_REASON_SUBS_TERMINATED**
The subscription is terminated.
Subscription Type Definitions
RvSipSubsStateChangeReason

RVSIP_SUBS_REASON_DIALOG_TERMINATION
The subscription dialog was terminated.

RVSIP_SUBS_REASONREFER_RCVD
A REFER request was received.

RVSIP_SUBS_REASONREFER_RCVD_WITH_REPLACE
A REFER request that contains a Replaces header was received.

RVSIP_SUBS_REASON_ILLEGALREFER_Msg
An illegal REFER request was received, which causes the subscription to terminate.
RvSipSubscriptionType

DESCRIPTION
Indicates whether or not the subscription represents a subscriber or notifier subscription.

SYNTAX
typedef enum{
   RVSIP_SUBS_TYPE_UNDEFINED = -1,
   RVSIP_SUBS_TYPE_SUBSCRIBER,
   RVSIP_SUBS_TYPE_NOTIFIER
}RvSipSubscriptionType;
RvSipSubsNotifyStatus

DESCRIPTION
An enumeration of the notification status (used in RvSipSubsNotifyEv()).

SYNTAX

typedef enum{
   RVSIP_SUBS_NOTIFY_STATUS_UNDEFINED = -1,
   RVSIP_SUBS_NOTIFY_STATUS_IDLE,
   RVSIP_SUBS_NOTIFY_STATUS_REQUEST_RCVD,
   RVSIP_SUBS_NOTIFY_STATUS_2XX_RCVD,
   RVSIP_SUBS_NOTIFY_STATUS_REJECT_RCVD,
   RVSIP_SUBS_NOTIFY_STATUS_REDIRECTED,
   RVSIP_SUBS_NOTIFY_STATUS_UNAUTHENTICATED,
   RVSIP_SUBS_NOTIFY_STATUS_TERMINATED,
   RVSIP_SUBS_NOTIFY_STATUS_MSG_SEND_FAILURE,
   RVSIP_SUBS_NOTIFY_STATUS_REQUEST_SENT,
   RVSIP_SUBS_NOTIFY_STATUS_FINAL_RESPONSE_SENT
}RvSipSubsNotifyStatus;

PARAMETERS

RVSIP_SUBS_NOTIFY_STATUS_IDLE
The initial status of a SIP Stack notification.

RVSIP_SUBS_NOTIFY_STATUS_REQUEST_RCVD

Upon receipt of a NOTIFY request by a subscriber subscription, the Stack notification informs the application of the REQUEST_RCVD status. At this stage, the application must decide whether to accept or reject the NOTIFY using the Subscription NOTIFY API.
Subscription Type Definitions
RvSipSubsNotifyStatus

**RVSIP_SUBS_NOTIFY_STATUS_REDIRECTED**

Upon receipt of a 3xx response to a NOTIFY request, the Stack *notification* informs the application of the REDIRECTED status. At this stage, the application may get needed information from the 3xx message. After this stage, the Stack *notification* is terminated and informs the application of the TERMINATED status.

**RVSIP_SUBS_NOTIFY_STATUS_UNAUTHENTICATED**

Upon receipt of a 401/407 response to a NOTIFY request, the *notification* informs the application of the UNAUTHENTICATED status. At this stage, the application may get needed information from the response message. After this stage, the Stack *notification* is terminated and informs the application of the TERMINATED status.

**RVSIP_SUBS_NOTIFY_STATUS_REJECT_RCVD**

Upon receipt of a non-2xx response to a NOTIFY request, the SIP Stack *notification* informs application of the REJECT_RCVD status. At this stage, the application may get needed information from the response message. After this stage, the Stack *notification* is terminated and informs the application of the TERMINATED status.

**RVSIP_SUBS_NOTIFY_STATUS_2XX_RCVD**

Upon receipt of a 2xx response to a NOTIFY request, the SIP Stack *notification* informs the application of the 2XX_RCVD status. At this stage, the application may get needed information from the 2xx message. After this stage, the Stack *notification* is terminated and informs the application of the TERMINATED status.

**RVSIP_SUBS_NOTIFY_STATUS_MSG_SEND_FAILURE**

A notifier that failed to send a NOTIFY request (receives a 503 response, transaction timeout or network error) informs the application of the MSG_SEND_FAILURE status. At this stage, the application must decide whether to re-send this message to the next IP address from the DNS list, or stop sending this message.

If the application stops sending the message, the SIP Stack *notification* is terminated and informs the application of the TERMINATED status. If the application re-sends the request to the next IP address, the SIP Stack uses the same *notification* to re-send the NOTIFY request.
Subscription Type Definitions
RvSipSubsNotifyStatus

RVSIP_SUBS_NOTIFY_STATUS_TERMINATED
The final status of which the SIP Stack notification gives notification. Upon receiving the Terminated status, you can no longer reference the notification.

RVSIP_SUBS_NOTIFY_STATUS_REQUEST_SENT
Upon sending a NOTIFY request, the SIP Stack notification informs the application of the REQUEST_SENT status. At this stage, SIP Stack notification waits for a response.

RVSIP_SUBS_NOTIFY_STATUS_FINAL_RESPONSE_SENT
Upon sending a NOTIFY response, the SIP Stack notification is informed of the FINAL_RESPONSE_SENT status. After this, the notification is terminated and informed of the TERMINATED status.
Subscription Type Definitions

RvSipSubsNotifyReason

**RvSipSubsNotifyReason**

**DESCRIPTION**

An enumeration of the reasons for the *notification* status (used in `RvSipSubsNotifyEv()`).

**SYNTAX**

```c
typedef enum{
    RVSIP_SUBS_NOTIFY_REASON_UNDEFINED = -1,
    RVSIP_SUBS_NOTIFY_REASON_SUBS_TERMINATED,
    RVSIP_SUBS_NOTIFY_REASON_TRANSC_TIMEOUT,
    RVSIP_SUBS_NOTIFY_REASON_TRANSC_ERROR,
    RVSIP_SUBS_NOTIFY_REASON_NETWORK_ERROR,
    RVSIP_SUBS_NOTIFY_REASON_503_RECEIVED,
    RVSIP_SUBS_NOTIFY_REASON_GIVE_UP_DNS,
    RVSIP_SUBS_NOTIFY_REASON_CONTINUE_DNS,
    RVSIP_SUBS_NOTIFY_REASON_APP_COMMAND,
    RVSIP_SUBS_NOTIFY_REASON_BAD_REQUEST_MESSAGE
    RVSIP_SUBS_NOTIFY_REASONREFER_NO_SUBSCRIPTION_STATE_HEADER,
    RVSIP_SUBS_NOTIFY_REASONREFER_NO_EXPIRES_IN_FIRST_NOTIFY
}RvSipSubsNotifyReason;
```

**PARAMETERS**

**RVSIP_SUBS_NOTIFY_REASON_UNDEFINED**

The reason is undefined. This reason is used when there is no need for additional information. For example, the status is NOTIFY_REQUEST_RECEIVED, and the reason is "undefined".

**RVSIP_SUBS_NOTIFY_REASON_SUBS_TERMINATED**

The *notification* is terminated because its *subscription* is terminated.
RvSIP_SUBS_NOTIFY_REASON_TRANSC_TIMEOUT
A timeout of the NOTIFY request.

RvSIP_SUBS_NOTIFY_REASON_TRANSC_ERROR
A transaction error occurred.

RvSIP_SUBS_NOTIFY_REASON_NETWORK_ERROR
A network error occurred.

RvSIP_SUBS_NOTIFY_REASON_503_RECEIVED
A 503 response was received.

RvSIP_SUBS_NOTIFY_REASON_GIVE_UP_DNS
The local application called RvSipSubsDNSGiveUp() function.

RvSIP_SUBS_NOTIFY_REASON_CONTINUE_DNS
The application called RvSipSubsDNSContinue() function.

RvSIP_SUBS_NOTIFY_REASON_APP_COMMAND
The reason is an application command. For example, the application called RvSipNotifyTerminate().

RvSIP_SUBS_NOTIFY_REASON REFER NO_SUBSCRIPTION_STATE HEADER
A NOTIFY request on a REFER subscription was received without a Subscription-state header in it. The recommended behavior is to accept this NOTIFY request and terminate the REFER subscription.

RvSIP_SUBS_NOTIFY_REASON REFER NO_EXPIRES_IN_FIRST NOTIFY
A first NOTIFY request on a REFER subscription was received without an expires parameter in it. The recommended behavior is to accept this NOTIFY request and terminate the REFER subscription.
RvSipSubsEventPackageType

DESCRIPTION

An enumeration of the type of a subscription event package.

SYNTAX

typedef enum {
    RVSIP_SUBS_EVENT_PACKAGE_TYPE_UNDEFINED = -1,
    RVSIP_SUBS_EVENT_PACKAGE_TYPEREFER,
    RVSIP_SUBS_EVENT_PACKAGE_TYPE_OTHER
} RvSipSubsEventPackageType;

PARAMETERS

RVSIP_SUBS_EVENT_PACKAGE_TYPE_UNDEFINED

The event package is undefined.

RVSIP_SUBS_EVENT_PACKAGE_TYPE_REFERER

The event package is a REFER package.

RVSIP_SUBS_EVENT_PACKAGE_TYPE_OTHER

The event package is an Other package.
**RvSipSubsReferNotifyReadyReason**

**DESCRIPTION**

An enumeration of the reasons for the `RvSipSubsReferNotifyReadyEv()` callback to be called. (The reasons identify the cause for sending a NOTIFY request).

**SYNTAX**

```c
typedef enum{
    RVSIP_SUBSREFER_NOTIFY_READY_REASON_UNDEFINED = -1,
    RVSIP_SUBSREFER_NOTIFY_READY_REASON_INITIAL_NOTIFY,
    RVSIP_SUBSREFER_NOTIFY_READY_REASON_1XX_RESPONSE_MSG_RCVD,
    RVSIP_SUBSREFER_NOTIFY_READY_REASON_FINAL_RESPONSE_MSG_RCVD,
    RVSIP_SUBSREFER_NOTIFY_READY_REASON_TIMEOUT,
    RVSIP_SUBSREFER_NOTIFY_READY_REASON_ERROR_TERMINATION,
    RVSIP_SUBSREFER_NOTIFY_READY_REASON_OBJ_TERMINATED,
    RVSIP_SUBSREFER_NOTIFY_READY_REASON_UNSUPPORTED_AUTH_PARAMS,
    RVSIP_SUBS_EXPIRES_VAL_LIMIT
}RvSipSubsReferNotifyReadyReason;
```

**PARAMETERS**

**RVSIP_SUBS_REFERER_NOTIFY_READY_REASON_UNDEFINED**

The reason is undefined.

**RVSIP_SUBS_REFERER_NOTIFY_READY_REASON_INITIAL_NOTIFY**

An initial NOTIFY should be sent after accepting the REFER request.

**RVSIP_SUBS_REFERER_NOTIFY_READY_REASON_1XX_RESPONSE_MSG_RCVD**

A 1xx response was received on the REFER subscription associated object.
Subscription Type Definitions
RvSipSubsReferNotifyReadyReason

RVSIP_SUBS REFER NOTIFY READY_REASON_FINAL_RESPONSE_MSG_RCVD
A final response was received on the REFER subscription associated object.

RVSIP_SUBS REFER NOTIFY READY_REASON_TIMEOUT
A timeout on the request of the REFER subscription associated object.

RVSIP_SUBS REFER NOTIFY READY_REASON_ERROR_TERMINATION
An error termination of the REFER subscription associated object (mostly because of a lack of resources).

RVSIP_SUBS REFER NOTIFY READY_REASON_OBJ_TERMINATED
The application terminated the object created by a REFER attempt.

RVSIP_SUBS REFER NOTIFY READY_REASON_UNSUPPORTED_AUTH_PARAMS
A CONNECT, SUBSCRIBE or REFER attempt received a 401/407 response with unsupported authentication parameters.

RVSIP_SUBS EXPIRES VAL LIMIT
The definition of the maximum expires value that the SIP Stack allows. The expires value is used in the Expires header of a SUBSCRIBE request, and in expires parameter in a Subscription-State header of a NOTIFY request.
RvSipSubsEvHandlers

DESCRIPTION

A structure with function pointers to the callback of the module. This structure is used to set the application callbacks in the RvSipSubsMgrSetEvHandlers() function.

SYNTAX

typedef struct{
    RvSipSubsCreatedEv
        pfnSubsCreatedEvHandler;
    RvSipSubsStateChangedEv
        pfnStateChangedEvHandler;
    RvSipSubsSubscriptionExpiredEv
        pfnSubsExpiredEvHandler;
    RvSipSubsExpirationAlertEv
        pfnExpirationAlertEvHandler;
    RvSipSubsNotifyEv
        pfnNotifyEvHandler;
    RvSipSubsAuthCredentialsFoundEv
        pfnAuthCredentialsFoundEvHandler;
    RvSipSubsAuthCompletedEv
        pfnAuthCompletedEvHandler;
    RvSipSubsMsgToSendEv
        pfnMsgToSendEvHandler;
    RvSipSubsMsgReceivedEv
        pfnMsgReceivedEvHandler;
    RvSipSubsNotifyCreatedEv
        pfnNotifyCreatedEvHandler;
    RvSipSubsOtherURLAddressFoundEv
        pfnOtherURLAddressFoundEvHandler;
    RvSipSubsFinalDestResolvedEv
        pfnFinalDestResolvedEvHandler;
    RvSipSubsReferNotifyReadyEv
        pfnReferNotifyReadyEvHandler;
}

Subscription Type Definitions

RvSipSubsEvHandlers

RvSipSubsNewConnInUseEv
  pfnNewConnInUseEvHandler;
RvSipSubsCreatedDueToForkingEv
  pfnSubsCreatedDueToForkingEvHandler;
}RvSipSubsEvHandlers;

PARAMETERS

pfnSubsCreatedEvHandler
Notifies the application of a new notification.

pfnStateChangedEvHandler
Notifies the application of a subscription state change.

pfnSubsExpiredEvHandler
Notifies the application that the subscription expired.

pfnExpirationAlertEvHandler
Notifies the application that the subscription is about to be expired.

pfnNotifyEvHandler
Notifies the application of a NOTIFY request or response.

pfnAuthCredentialsFoundEvHandler
Notifies the application of new credentials for server authentication.

pfnAuthCompletedEvHandler
Notifies the application that the server authentication procedure was completed (with success or failure).

pfnMsgToSendEvHandler
Notifies the application that a new message is going to be sent.

pfnMsgReceivedEvHandler
Notifies the application that a new message was received.
Subscription Type Definitions

**pfnNotifyCreatedEvHandler**
Notifies the application of a new *notification*.

**pfnOtherURLAddressFoundEvHandler**
Notifies the application that another URL address was found and has to be converted to a SIP URL address.

**pfnFinalDestResolvedEvHandler**
Notifies that the *subscription* 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. The application can use the Transaction API function to get the destination address.

**pfnReferNotifyReadyEvHandler**
Indicates that a REFER notifier *subscription* is ready to send a NOTIFY request.

**pfnNewConnInUseEvHandler**
Notifies that the *subscription* is now using a new *connection*. The *connection* can be a totally new *connection* or a suitable *connection* that was found in the hash.

**pfnSubsCreatedDueToForkingEvHandler**
Notifies the application about a new *subscription* that was created as a result of the arrival of a forked NOTIFY message. If this parameter is not set, the NOTIFY will be rejected with a 481 response and a *subscription* will not be created.
Subscription Callback Functions

The Subscriptions callback functions are:

- RvSipSubsCreatedEv()
- RvSipSubsStateChangedEv()
- RvSipSubsSubscriptionExpiredEv()
- RvSipSubsExpirationAlertEv()
RvSipSubsCreatedEv()

DESCRIPTION
Notifies the application that a new subscription was created, and exchanges handles with the application. If the subscription is related to an existing call-leg, the hCallLeg and hAppCallLeg parameters contain the call-leg handles. If the subscription is an independent subscription, both hCallLeg and hAppCallLeg contain NULL values.

SYNTAX

typedef void (RVCALLCONV * RvSipSubsCreatedEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipCallLegHandle hCallLeg,
    IN RvSipAppCallLegHandle hAppCallLeg,
    OUT RvSipAppSubsHandle *phAppSubs);

PARAMETERS

hSubs
The new SIP Stack subscription handle.

hCallLeg
The handle to the related call-leg. NULL if this is a subscription outside of a call-leg.

hAppCallLeg
The handle to the related application call-leg. NULL if this is a subscription outside of a call-leg.

phAppSubs
The application handle for this subscription.

RETURN VALUES
None.
Subscription Callback Functions
RvSipSubsStateChangedEv()

RvSipSubsStateChangedEv()

DESCRIPTION
Notifies the application of a subscription state change. For each state change the new state is supplied with the reason for the state change.

SYNTAX

typedef void (RVCALLCONV * RvSipSubsStateChangedEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs,
    IN RvSipSubsState eState,
    IN RvSipSubsStateChangedReason eReason);

PARAMETERS

hSubs
The SIP Stack subscription handle.

hAppSubs
The application handle for this subscription.

eState
The new subscription state.

eReason
The reason for the state change.

RETURN VALUES
None.
**RvSipSubsSubscriptionExpiredEv()**

**DESCRIPTION**

Notifies the application of a *subscription* expiration. A Notifier application should send a NOTIFY request with a “Subscription-State:Terminated” header to change to the Subs-Terminating state. A Subscriber application can try sending a refresh SUBSCRIBE request.

**SYNTAX**

```c
typedef void (RVCALLCONV * RvSipSubsSubscriptionExpiredEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs);
```

**PARAMETERS**

- **hSubs**
  The SIP Stack *subscription* handle.

- **hAppSubs**
  The application handle for this *subscription*.

**RETURN VALUES**

None.
Subscription Callback Functions
RvSipSubsExpirationAlertEv()

RvSipSubsExpirationAlertEv()

DESCRIPTION
Alerts the application that the subscription timer is about to expire. The application should use this callback to send a refresh when the subscription state is ACTIVE. The time interval of the subscription alert timer is set with the “subsAlertTimer” configuration parameter. (If subsAlertTimer is set to 0, or if subsAutoRefresh is set to RV_TRUE, this callback function will not be called).

SYNTAX

typedef void (RVCALLCONV * RvSipSubsExpirationAlertEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs);

PARAMETERS

hSubs
The SIP Stack subscription handle.

hAppSubs
The application handle for this subscription.

RETURN VALUES
None.
NOTIFICATION
CALLBACK
FUNCTIONS

The Notification callback function is:

- RvSipSubsNotifyCreatedEv()
- RvSipSubsNotifyEv()
- RvSipSubsCreatedDueToForkingEv()
**RvSipSubsNotifyCreatedEv()**

**DESCRIPTION**
Indicates that a new notification was created, and exchanges handles with the application.

**SYNTAX**
```c
typedef void (RVCALLCONV *RvSipSubsNotifyCreatedEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs,
    IN RvSipNotifyHandle hNotification,
    OUT RvSipAppNotifyHandle *phAppNotification);
```

**PARAMETERS**

**hSubs**
The SIP Stack subscription handle

**hAppSubs**
The application handle for this subscription.

**hNotification**
The newly created notification handle.

**phAppNotification**
The application handle for this notification.

**RETURN VALUES**
None.
RvSipSubsNotifyEv()

DESCRIPTION

Notifies the application of the notify status. For a subscriber, this callback gives the received NOTIFY message and its related notification handle. The application can get all the notification information from the message with this callback. The message is destructed when returning from this callback.

For a notifier, this callback informs the application of a response message that was received on a NOTIFY request and supplies the response message. When returning from this callback function, the Stack notification is destructed. For both a subscriber and a notifier, this function notifies the application about the termination of the notification.

SYNTAX

typedef void (RVCALLCONV * RvSipSubsNotifyEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs,
    IN RvSipNotifyHandle hNotification,
    IN RvSipAppNotifyHandle hAppNotification,
    IN RvSipSubsNotifyStatus eNotifyStatus,
    IN RvSipSubsNotifyReason eNotifyReason,
    IN RvSipMsgHandle hNotifyMsg);

PARAMETERS

hSubs

The SIP Stack subscription handle.

hAppSubs

The application handle for this subscription.

hNotification

The newly created notification handle.
Notification Callback Functions
RvSipSubsNotifyEv()

**hAppNotification**
The application handle for this *notification*.

**eNotifyStatus**
The status of the *notification*.

**eNotifyReason**
The reason for the indicated status.

**hNotifyMsg**
The received message (notify, request or response).

**RETURN VALUES**
None.
**RvSipSubsCreatedDueToForkingEv()**

**DESCRIPTION**

Notifies the application of the creation of a forked subscription. A forked subscription is created when a forked NOTIFY request arrives. For more information, see RFC 3265 (sections 3.3.3 and 4.4.9) for the criteria of the forking determination.

The application can decide to reject the subscription creation on this callback. It should set the pRejectStatus parameter to the error code, which will be set in the response to the NOTIFY request. A forked subscription will not be created if the original subscription (the subscription that generated the SUBSCRIBE request) is configured to disable forking. For more information, see the RvSipSubsSetForkingEnabledFlag() function.

If the application does not set the callback, the NOTIFY will be rejected with a 481 response according to RFC 3265, and the transaction will be closed. The application should not use any API functions which have an affect on the subscription state in this callback (such as RvSipSubsTerminate()). It is recommended that the application disable forking for the event package of the REFER type. This means that if a subscriber subscription is created as a result of an incoming REFER request with the SUBSCRIBE method (the handle to such an object is returned by RvSipSubsReferAccept()), the application should set the “forking enabled” flag for the object to RV_FALSE (by means of RvSipSubsSetForkingEnabledFlag()).

**SYNTAX**

```c
typedef RvStatus (RVCALLCONV
*RvSipSubsCreatedDueToForkingEv)(
    IN    RvSipSubsHandle hSubs,
    INOUT RvInt32 *pExpires,
    OUT   RvSipAppSubsHandle *phAppSubs,
    OUT   RvUint16 *pRejectStatus);
```

**PARAMETERS**

- **hSubs**

  The handle to the forked subscription that was created.
Notification Callback Functions

**pExpires**

**INPUT:** A pointer to the expiration value of the subscription. This value should be set in seconds. If the value was not determined, it will be set to –1 (it was not set in the original subscription). In this case, the application should supply the value to limit the life cycle of the forked subscription. The RvSipSubsUpdateSubscriptionTimer() function can also be used for this purpose. On expiration, the expiration callback will be called for the forked subscription.

**OUTPUT:** A pointer to the memory where the application can set the expiration value.

**phAppSubs**

A pointer to the memory where the application can store the handle to the application subscription.

**pRejectStatus**

If the application decides to terminate this subscription, it should set the pointer to point to a positive integer, representing the error code. In this case, the SIP Stack will respond to the forked NOTIFY with the error code and will free the subscription and all the subscription-related resources.

**RETURN VALUES**

Any. (The returned status is currently ignored.)
Authentication Callback Functions

The Authentication callback functions are:

- `RvSipSubsAuthCredentialsFoundEv()`
- `RvSipSubsAuthCompletedEv()`
Authentication Callback Functions
RvSipSubsAuthCredentialsFoundEv()

**RvSipSubsAuthCredentialsFoundEv()**

**DESCRIPTION**

Notifies the application that credentials were found in the SUBSCRIBE or NOTIFY Request message. This callback is called only if the application started the authentication procedure by calling `RvSipSubsAuthBegin()`. The callback supplies the application with the Authorization header that contains these credentials. At this stage, the application should use the `RvSipSubsAuthProceed()` function.

This callback also indicates whether the SIP Stack is capable of verifying the credentials that were found. Whenever the SIP Stack does not support the credentials (for example, if the algorithm is not MD5) the application can verify the credentials itself or instruct the SIP Stack to continue to the next header.

The authentication procedure can be done for a SUBSCRIBE or a NOTIFY request. For a SUBSCRIBE request, `hNotification` is NULL. Otherwise, `hNotification` is the handle to the notification that received the NOTIFY request.

**Note**  You must get the credentials from the given Authorization header synchronically, or you should copy the header to use it a-synchronously. When you have the credentials, you can check the validity and continue with `RvSipAuthProceed` a-synchronously.

**SYNTAX**

```c
typedef void (RVCALLCONV * RvSipSubsAuthCredentialsFoundEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs,
    IN RvSipNotifyHandle hNotification,
    IN RvSipAuthorizationHeaderHandle hAuthorization,
    IN RvBool bCredentialsSupported);
```

**PARAMETERS**

- **hSubs**
  The SIP Stack `subscription` handle.
hAppSubs
The application handle for this subscription.

hNotification
The handle to the notification that received the request.

hAuthorization
The handle to the Authorization header that contains the credentials.

bCredentialsSupported
Determines whether or not the SIP Stack supports the Authorization header credentials.

RETURN VALUES
None.
Authentication Callback Functions
RvSipSubsAuthCompletedEv()

RvSipSubsAuthCompletedEv()

DESCRIPTION
Called when the authentication procedure is completed. If the procedure is completed because the correct Authorization header was found, bAuthSucceed is RV_TRUE. If the procedure is completed because there are no more Authorization headers to check, or because the application ordered to stop searching for the correct header, bAuthSucceed is RV_FALSE. The authentication procedure can be done for a SUBSCRIBE or a NOTIFY request. For a SUBSCRIBE request, hNotification is NULL. Otherwise, hNotification is the handle to the notification that received the NOTIFY request.

SYNTAX
typedef void (RVCALLCONV * RvSipSubsAuthCompletedEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs,
    IN RvSipNotifyHandle hNotification,
    IN RvBool bAuthSucceed);

PARAMETERS

hSubs
The handle to the subscription that received the SUBSCRIBE request. For a NOTIFY request, this parameter is the handle to the notification-related subscription.

hAppSubs
The application handle for this subscription.

hNotification
The handle to the notification that received the NOTIFY request. NULL for a SUBSCRIBE request.

bAuthSucceed
RV_TRUE if the correct Authorization header was found. Otherwise, RV_FALSE.
RETURN VALUES

None.
Message Callback Functions

The Message callback functions are:

- RvSipSubsMsgToSendEv()
- RvSipSubsMsgReceivedEv()
**RvSipSubsMsgToSendEv()**

**DESCRIPTION**

Notifies the application that a *subscription*-related outgoing message is about to be sent (SUBSCRIBE or NOTIFY requests and responses).

**SYNTAX**

```c
typedef RvStatus (RVCALLCONV * RvSipSubsMsgToSendEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs,
    IN RvSipNotifyHandle hNotify,
    IN RvSipAppNotifyHandle hAppNotify,
    IN RvSipMsgHandle hMsg);
```

**PARAMETERS**

- **hSubs**
  The SIP Stack *subscription* handle

- **hAppSubs**
  The application handle for this *subscription*.

- **hNotify**
  The notify object handle. (Relevant only for a NOTIFY message.)

- **hAppNotify**
  The application notify object handle. (Relevant only for a NOTIFY message.)

- **hMsg**
  The handle to the outgoing message.

**RETURN VALUES**

Returns `RvStatus`.

(The returned status is ignored in the current version of the Stack.)
Message Callback Functions

RvSipSubsMsgReceivedEv()

**RvSipSubsMsgReceivedEv()**

**DESCRIPTION**

Notifies the application that a *subscription*-related incoming message has been received (SUBSCRIBE or NOTIFY requests and responses).

**SYNTAX**

```c
typedef RvStatus (RVCALLCONV * RvSipSubsMsgReceivedEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs,
    IN RvSipNotifyHandle hNotify,
    IN RvSipAppNotifyHandle hAppNotify,
    IN RvSipMsgHandle hMsg);
```

**PARAMETERS**

*hSubs*

The SIP Stack *subscription* handle.

*hAppSubs*

The application handle for this *subscription*.

*hNotify*

The notify object handle. (Relevant only for a NOTIFY message.)

*hAppNotify*

The application notify object handle. (Relevant only for a NOTIFY message.)

*hMsg*

The handle to the incoming message.

**RETURN VALUES**

Returns RvStatus.

(The returned status is ignored in the current version of the Stack.)
The REFER callback functions are:

- RvSipSubsReferNotifyReadyEv()
REFER Callback Functions

RvSipSubsReferNotifyReadyEv()

**DESCRIPTION**

Notifies the application that a REFER notifier *subscription* is ready to send a NOTIFY request. This callback is called after accepting a REFER request, and for every incoming provisional or final response for the related object establishment attempt. When this callback is called, application should create and send a new NOTIFY request, with the status of the establish attempt of the Refer-related object. The response code parameter indicates the response code that should be set in the body of the NOTIFY message. This callback also provides the incoming response message of the establishment attempt. (Note that the message does not always exist. For example, in the case of timeout termination, this parameter will be NULL).

**SYNTAX**

```c
typedef void (RVCALLCONV * RvSipSubsReferNotifyReadyEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs,
    IN RvSipSubsReferNotifyReadyReason eReason,
    IN RvInt16 responseCode,
    IN RvSipMsgHandle hResponseMsg);
```

**PARAMETERS**

- **hSubs**
  The SIP Stack *subscription* handle.

- **hAppSubs**
  The application handle for this *subscription*.

- **eReason**
  The reason for a NOTIFY request to be sent.

- **responseCode**
  The response code that should be set in the NOTIFY message body.
hResponseMsg

The message that was received on the Refer-related object (provisional or final response).

RETURN VALUES

None.
General Callback Functions

The Other callback functions are:

- RvSipSubsOtherURLAddressFoundEv()
- RvSipSubsFinalDestResolvedEv()
- RvSipSubsNewConnInUseEv()
- RvSipSubsCreatedDueToForkingEv()
RvSipSubsOtherURLAddressFoundEv()

DESCRIPTION
Notifies the application that an additional URL address (a URL address that is currently not supported by the SIP Stack) was found and has to be converted to a known SIP URL address.

SYNTAX

```c
typedef RvStatus
  (RVCALLCONV * RvSipSubsOtherURLAddressFoundEv)(
   IN RvSipSubsHandle hSubs,
   IN RvSipAppSubsHandle AppSubs,
   IN RvSipNotifyHandle hNotify,
   IN RvSipAppNotifyHandle hAppNotify,
   IN RvSipMsgHandle hMsg,
   IN RvSipAddressHandle hAddress,
   OUT RvSipAddressHandle hSipURLAddress,
   OUT RvBool *bAddressResolved);
```

PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hCallLeg</td>
<td>The SIP Stack call-leg handle.</td>
</tr>
<tr>
<td>hAppCallLeg</td>
<td>The application handle for this call-leg.</td>
</tr>
<tr>
<td>hTransc</td>
<td>The transaction handle.</td>
</tr>
<tr>
<td>hAppTransc</td>
<td>The application handle for this transaction.</td>
</tr>
<tr>
<td>hMsg</td>
<td>The message that includes the other URL address.</td>
</tr>
</tbody>
</table>
General Callback Functions
RvSipSubsOtherURLAddressFoundEv()

**hAddress**
The handle to the unsupported address that is to be converted.

**hSipURLAddress**
The handle to the known SIP URL address.

**bAddressResolved**
Indicates whether the address resolution was a success or a failure.

**RETURN VALUES**
Returns RvStatus.
RvSipSubsFinalDestResolvedEv()

DESCRIPTION

Notifies the application that the subscription 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. When this callback is called, the application can query the transaction about the destination address using the RvSipTransactionGetCurrentDestAddress() function. If the application wishes, it can update the “sent-by” part of the top-most Via header. The application must not update the branch parameter.

SYNTAX

typedef RvStatus
(RVCALLCONV * RvSipSubsFinalDestResolvedEv)(
    IN RvSipSubsHandle hSubs,
    IN RvSipAppSubsHandle hAppSubs,
    IN RvSipNotifyHandle hNotify,
    IN RvSipAppNotifyHandle hAppNotify,
    IN RvSipTranscHandle hTransc,
    IN RvSipMsgHandle hMsgToSend);

PARAMETERS

hSubs
The SIP Stack subscription handle.

hAppSubs
The application handle for this subscription.

hNotify
The NOTIFY object handle (relevant only for a NOTIFY request or response).

hAppNotify
The application NOTIFY object handle (relevant only for a NOTIFY request or response).
General Callback Functions
RvSipSubsFinalDestResolvedEv()

hTransc
The transaction handle.

hMsgToSend
The handle to the outgoing message.

RETURN VALUES
Returns RvStatus.
If the application returns a value other than RV_OK, the message will not be sent. The transaction will terminate with an error and the subscription will behave according to its processing rules for transaction error termination.
RvSipSubsNewConnInUseEv()

DESCRIPTION
Notifies the application that the subscription is now using a new connection. The connection can be a totally new connection or a suitable connection that was found in the hash.

SYNTAX

typedef RvStatus (RVCALLCONV *RvSipSubsNewConnInUseEv) (  
  IN RvSipSubsHandle hSubs,  
  IN RvSipAppSubsHandle hAppSubs,  
  IN RvSipNotifyHandle hNotify,  
  IN RvSipAppNotifyHandle hAppNotify,  
  IN RvSipTransportConnectionHandle hConn,  
  IN RvBool bNewConnCreated);

PARAMETERS

hSubs
The SIP Stack subscription handle.

hAppSubs
The application handle for this subscription.

hNotify
The notify object handle (relevant only for NOTIFY request or response).

hAppNotify
The application notify object handle (relevant only for NOTIFY request or response).

hConn
The connection handle.
General Callback Functions
RvSipSsubsNewConnInUseEv()

**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 RV_OK. (The returned status is ignored in the current version of the SIP Stack.)
RvSipSubsCreatedDueToForkingEv()

DESCRIPTION

Notifies the application about the creation of a forked subscription. A forked subscription is created as the result of the arrival of a forked NOTIFY message. For information about forking determination criteria, see sections 3.3.3 and 4.4.9 of RFC 3265.

Using this callback, the application can decide to reject the creation of the subscription. To do so, it should set the pRejectStatus parameter to the error code, which will be set in response to the NOTIFY message. The forked subscription will not be created if the original subscription (the one that generated the SUBSCRIBE request) is configured to disable forking. See the RvSipSubsSetForkingEnabledFlag() function.

If the callback is not set by the application, the NOTIFY message will be rejected with a 481 response according RFC 3265, and the transaction will be closed. The application should not call the Subscription Layer API from this callback. Since it may affect the state of the Subscription (for example, RvSipSubsTerminate()).

It is recommended that the application disable forking for the event package of the REFER type. This means that if a subscriber subscription is created as the result of an incoming REFER request with a SUBSCRIBE method (the handle to such object is returned by RvSipSubsReferAccept()), the application should set the “Forking Enabled” flag for the object to RV_FALSE (by means of the RvSipSubsSetForkingEnabledFlag() function).

SYNTAX

typedef RvStatus (RVCALLCONV
*RvSipSubsCreatedDueToForkingEv)(
    IN    RvSipSubsHandle    hSubs,
    INOUT RvInt32           *pExpires,
    OUT   RvSipAppSubsHandle *phAppSubs,
    OUT   RvUint16           *pRejectStatus);

PARAMETERS

hSubs

The handle to the created forked subscription.
General Callback Functions
RvSipSubsCreatedDueToForkingEv()

pExpires

IN: A pointer to the expiration value of the subscription. This value should be set in seconds. If the value was not determined, it will be set to –1. The value cannot be determined if it was not set in the original subscription. In this case, the application should supply the value to limit the life cycle of the forked subscription. The RvSipSubsUpdateSubscriptionTimer() function can be also used for this purpose. On expiration, the expiration callback will be called for the forked subscription.

OUT: A pointer to the memory where the application can set the expiration value.

phAppSubs

The handle that the application sets for the subscription.

pRejectStatus

If the application decides to terminate this subscription, it should set the pointer to point to a positive integer, representing the error code. In this case, the SIP Stack will respond to the forked NOTIFY with error code and will free the subscription and all the subscription related-resources.

RETURN VALUES

Returns any value. (The returned status is ignored in the current release.)
TRANSPORT API TYPE DEFINITIONS

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:

- Transport Handle Type Definitions
- Transport Type Definitions
- Transport DNS Type Definitions
- Transport Callback Functions
- Transport Object Event Handler
Transport Handle Type Definitions

The handle type definitions are:

- RvSipTransportMgrHandle
- RvSipAppTransportMgrHandle
- RvSipTransportConnectionHandle
- RvSipTransportConnectionOwnerHandle
- RvSipTransportConnectionAppHandle
- RvSipTransportTlsEngineHandle
- RvSipTransportLocalAddrHandle
- RvSipTransportLocalAddrAppHandle
- RvSipTransportTlsCertificate
**RvSipTransportMgrHandle**

**DESCRIPTION**

The declaration of a handle to the *TransportMgr*.

**SYNTAX**

```c
RV_DECLARE_HANDLE(RvSipTransportMgrHandle);
```

**PARAMETERS**

None.

**RETURN VALUES**

None.
RvSipAppTransportMgrHandle

**DESCRIPTION**

The declaration of an application handle to a TransportMgr. This handle can be used as a context that the application can hold in the TransportMgr.

**SYNTAX**

RV_DECLARE_HANDLE(RvSipAppTransportMgrHandle);

**PARAMETERS**

None.

**RETURN VALUES**

None.
RvSipTransportConnectionHandle

DESCRIPTION
The declaration of handle to a connection. A connection is used for TCP or 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. 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.
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);
RvSipTransportLocalAddrAppHandle

DESCRIPTION
An application handle to the local address object that the application can set and get from the connection. A local address can hold only one application handle.

SYNTAX
RV_DECLARE_HANDLE(RvSipTransportLocalAddrAppHandle);

PARAMETERS
None.

RETURN VALUES
None.
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.
Transport Type Definitions

**Transport Type Definitions**

The Transport type definitions are:

- RvSipTransportAddressType
- RvSipTransportPersistencyLevel
- RvSipTransportConnectionCfg
- RvSipTransportConnectionState
- RvSipTransportConnectionTlsState
- RvSipTransportConnectionStatus
- RvSipTransportConnectionTlsStatus
- RvSipTransportConnectionStateChangedReason
- RvSipTransportPrivateKeyType
- RvSipTransportTlsMethod
- RvSipTransportMsgAddrCfg
- RvSipTransportBsAction
- RvSipTransportMgrEvHandlers
- RvSipTransportConnectionEvHandlers
- RvSipTransportTlsEngineCfg
- RvSipTransportTlsHandshakeSide
- RvSipTransportAddr
- RvSipTransportOutboundProxyCfg
- 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. Undefined means that there is no persistency of connections. One connection is used for a request and its response. After the response is sent, the connection is closed.

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 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 transactions 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).
Transport Type Definitions
RvSipTransportConnectionCfg

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{
    RvSipTransport eTransportType;
    RvChar* strLocalIp;
    RvUint16 localPort;
    RvChar* strDestIp;
    RvUint16 destPort;
    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. Zero 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.
Transport Type Definitions
RvSipTransportConnectionState

RvSipTransportConnectionState

DESCRIPTION

Represents the state of a connection. The state changes are reported using the RvSipTransportConnectionStateChangedEv() callback function with a state change reason.

Note  For a detailed description of the connection states, see the Working with the Transport Layer chapter in the HP-UX C SIP Stack Programmer's Guide.

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;
**RvSipTransportConnectionTlsState**

**DESCRIPTION**

The TLS state of the connection.

**Note**  For a detailed description of the TLS connection states, see the Working with the Transport Layer chapter in the HP-UX C SIP Stack Programmer’s Guide.

**SYNTAX**

```c
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_SEQUENCE_STARTED,
    RVSIP_TRANSPORT_CONN_TLS_STATE_CONNECTED,
    RVSIP_TRANSPORT_CONN_TLS_STATE_TERMINATED
}RvSipTransportConnectionTlsState;
```

**PARAMETERS**

**RVSIP_TRANSPORT_CONN_TLS_STATE_UNDEFINED**

A TLS sequence was not 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 is ready to start the TLS handshake.

**RVSIP_TRANSPORT_CONN_TLS_STATE_HANDSHAKE_STARTED**

The connection is performing a handshake.
Transport Type Definitions
RvSipTransportConnectionTlsState

**RVSIP_TRANSPORT_CONN_TLS_STATE_HANDSHAKE_FAILED**
The TLS handshake failed. Data cannot be transmitted on the connection.

**RVSIP_TRANSPORT_CONN_TLS_STATE_CLOSE_SEQUENCE_STARTED**
The connection received or sent a close request but the connection did not close yet.

**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, the connection may not 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**

Represents the status of a connection. The SIP Stack reports the status to the application using the `RvSipTransportConnectionTlsStatusEv()` callback.

**SYNTAX**

```c
typedef enum{
    RVSIP_TRANSPORT_CONN_TLS_STATUS_UNDEFINED = 1,
    RVSIP_TRANSPORT_CONN_TLS_STATUS_HANDSHAKE_PROGRESS
}RvSipTransportConnectionState;
```

**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_ASSERTION_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_V23 =
    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 in [%sid format. If NULL is supplied, a default destination IP is chosen.

destPort
The destination 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 RvSipTransportBadSyntaxMsgEv() and RvSipTransportBadSyntaxStartLineMsgEv() 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, discards the bad-syntax message. (The same behavior as 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;
} RvSipTransportMgrEvHandlers;
Transport Type Definitions
RvSipTransportMgrEvHandlers

#if defined(RV_SSL_SESSION_STATUS)
    RvSipTransportConnectionTlsStatusEv
        pfnEvConnTlsStatus;
#endif /*RV_SSL_SESSION_STATUS*/
    RvSipTransportMsgReceivedExtEv
        pfnEvMsgRecvdExt;
    RvSipTransportConnectionDataReceivedEv
        pfnEvConnDataReceived;
    RvSipTransportConnectionServerReuseEv
        pfnEvConnServerReuse;
}RvSipTransportMgrEvHandlers;

PARAMETERS

pfnEvMsgToSend
Notifies the application that a message is about to be sent. Allows the application to decide whether or not to continue sending this message.

pfnEvMsgReceived
Notifies the application that a message was received. Allows the application to decide whether or not to continue processing this message.

pfnThreadError
Notifies the application of a thread that was exited unexpectedly.

pfnEvBadSyntaxMsg
Notifies the application that a bad-syntax message was received.

pfnEvBadSyntaxStartLineMsg
Notifies the application that a message with a bad-syntax start-line was received.

pfnEvTlsStateChanged
Notifies the application on TLS connection state changes.
**Transport Type Definitions**

**RvSipTransportMgrEvHandlers**

- **pfnEvTlsPostConnectionAssertion**
  If set to a value other the NULL, this parameter will let the application override post connection assertions that fail.

- **pfnEvTlsSeqStarted**
  Notifies the application of 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 the application about the creation of a connection for an incoming TCP connection.

- **pfnEvConnParserResult**
  Notifies the application of the result of the parsing of an incoming message.

- **pfnEvConnStateChanged**
  Notifies the application about an incoming connection state change.

- **pfnEvConnTlsStatus**
  Notifies the application on security events.

- **pfnEvMsgRecvdExt**
  Notifies application about receiving a SIP message.

- **pfnEvConnDataReceived**
  Notifies application about receiving data on the connection.
Transport Type Definitions
RvSipTransportMgrEvHandlers

\textbf{pfnEvConnServerReuse}

Notifies the application about a server \textit{connection} that can be reused.
**RvSipTransportConnectionEvHandlers**

**DESCRIPTION**
A structure with function pointers to the connection callbacks. You should supply this structure for every new connection.

**SYNTAX**
```c
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.
RvSipTransportTlsEngineCfg

certLen
The size of the certificate.

certDepth
The maximum length of a chain of certificates before it is considered invalid.
Transport Type Definitions
RvSipTransportTlsHandshakeSide

---

**RvSipTransportTlsHandshakeSide**

**DESCRIPTION**

Determines the side of the TLS handshake to assume. Default means that the TCP client will act as a TLS client and the TCP server will act as a TLS server. Default is the recommended way of working.

**SYNTAX**

```c
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 containing parameters of the address that the SIP Stack can use for network communication.

SYNTAX
```c
typedef struct{
    RvSipTransport eTransportType;
    RvSipTransportAddressType eAddrType;
    RvUint16 port;
    RvChar strIP[RVSIP_TRANSPORT_LEN_STRING_IP];
    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 address. (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 the configurations needed to set the outbound proxy details. These will be used in case of outgoing messages toward a proxy.

SYNTAX

typedef struct{
   RvChar *strIpAddress;
   RvInt32 port;
   RvChar *strHostName;
   RvSipTransport eTransport;
}RvSipTransportOutboundProxyCfg;

PARAMETERS

strIpAddress
Indicates the IP address of an outbound proxy that the SIP Stack uses.
Default Value: NULL. No outbound proxy.

port
Indicates 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
Indicates the transport of the outbound proxy that the SIP Stack uses.
Default Value: RVSIP_TRANSPORT_UNDEFINED
RvSipTransportRcvdMsgDetails

DESCRIPTION
A structure containing all the data that is related to a received message.

SYNTAX
```c
typedef struct{
    RvSipTransportConnectionHandle hConnInfo;
    RvSipTransportLocalAddrHandle hLocalAddr;
    RvSipTransportAddr recvFromAddr;
    RvSipCompType eCompression; Not Supported
    RvSipTransport BsAction 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).

SYNTAX
typedef enum{
   RVSIP_TRANSPORT_CONN_TYPE_UNDEFINED,
   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_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.
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
The callback function to notify the owner about the connection status.

pfnConnStausEvHandler
The callback function to notify the owner about the connection status, such as ERROR.
Transport DNS Type Definitions

The Transport DNS type definitions are:

- RvSipTransportDNSSRVEElement
- 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 RFC 1035 (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[16];
    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
RvSipTransportDNSIPElement

The Transport callback functions are:

- RvSipTransportConnectionStateChangedEv()
- RvSipTransportConnectionStatusEv()
- RvSipTransportVerifyCertificateEv()
- RvSipTransportConnectionTlsSequenceStartedEv()
- RvSipTransportConnectionTlsStateChangedEv()
- RvSipTransportConnectionTlsPostConnectionAssertionEv()
- RvSipTransportMsgThreadError()
- RvSipTransportBadSyntaxMsgEv()
- RvSipTransportBadSyntaxStartLineMsgEv()
- RvSipTransportConnectionCreatedEv()
- RvSipTransportBufferReceivedEv()
- RvSipTransportBufferToSendEv()
- RvSipTransportConnectionParserResultEv()
- RvSipTransportMsgToSendEv()
- RvSipTransportMsgReceivedEv()
- RvSipTransportMsgReceivedExtEv()
- RvSipTransportConnectionTlsStatusEv()
- RvSipTransportMsgReceivedExtEv()
- RvSipTransportConnectionDataReceivedEv()
- RvSipTransportConnectionServerReuseEv()
RvSipTransportConnectionStateChangedEv()

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 *RvSipTransportConnectionStateChangedEv) (
    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.
Transport Callback Functions
RvSipTransportConnectionStateChangedEv()

**eReason**
The 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.)
RvSipTransportConnectionStatusEv()

DESCRIPTION
The connection notifies about events that do not effect the connection state using the connection status callback. If, for example, there was an error in the connection, the connection will notify the application with RVSIP_TRANSPORT_CONN_STATUS_ERROR. The connection will then disconnect with the RVSIP_TRANSPORT_CONN_REASON_ERROR reason.

Note You do not have to register to this callback if you do not want to get connection statuses.

SYNTAX
typedef RvStatus
(RVCALLCONV *RvSipTransportConnectionStatusEv) ( 
    IN RvSipTransportConnectionHandle hConn,
    IN RvSipTransportConnectionOwnerHandle hOwner,
    IN RvSipTransportConnectionStatus eStatus,
    IN void* pInfo);

PARAMETERS
hConn
The connection handle

hOwner
The handle to the connection owner.

eStatus
The connection status.

RETURN VALUES
Returns RvStatus. (The returned status is ignored in the current SIP Stack version.)
Transport Callback Functions
RvSipTransportVerifyCertificateEv()

**RvSipTransportVerifyCertificateEv()**

**DESCRIPTION**
Notifies the application whenever a certificate needs to be processed. If you wish to leave the decision of the SIP Stack regarding a certificate, return the `prevError` parameter as the return value.

**SYNTAX**
```c
typedef RvInt32 (*RvSipTransportVerifyCertificateEv)(
    IN RvInt32 prevError,
    IN RvSipTransportTlsCertificate certificate);
```

**PARAMETERS**
- **prevError**
  An error previously detected by the SIP Stack. A positive number indicates that the certificate is OK.

- **certificate**
  The certificate for which this callback is called. This certificate can be encoded using the `RvSipTransportTlsEncodeCert()` function.

**RETURN VALUES**
Returns RvInt32.
**RvSipTransportConnectionTlsSequenceStartedEv()**

**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**

```c
typedef void (RVCALLCONV
*RvSipTransportConnectionTlsSequenceStartedEv)(
    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.
RvSipTransportConnectionTlsStateChangedEv()

**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**
```c
typedef RvStatus
(RVCALLCONV * RvSipTransportConnectionTlsStateChangedEv)(
    IN RvSipTransportConnectionHandle hConnection,
    IN RvSipTransportConnectionAppHandle hAppConnection,
    IN RvSipTransportConnectionTlsState eState,
    IN RvSipTransportConnectionStateChangedReason eReason);
```

**PARAMETERS**
- **hConnection**
The handle to the connection that changed the TLS state.
- **hAppConnection**
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.)
RvSipTransportConnectionTlsPostConnectionAssertionEv()

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

PARAMETERS

hConnection

The handle to the connection that changed the TLS state.

hAppConnection

The application handle to 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.
Transport Callback Functions
RvSipTransportConnectionTlsPostConnectionAssertionEv()

pbAsserted

RV_TRUE if the connection was asserted successfully. RV_FALSE if the assertion failed. In this case, the connection will be terminated automatically.
RvSipTransportMsgThreadError()

DESCRIPTION
Notifies that one of the SIP Stack processing threads terminated unexpectedly. Multithreaded applications can register to this callback.

SYNTAX
typedef void (RVCALLCONV * RvSipTransportMsgThreadError)(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvStatus error);

PARAMETERS
hTransportMgr
The handle to the TransportMgr.

error
The RV_ERROR_NUM_OF_THREADS_DECREASED error.

RETURN VALUES
None.
Transport Callback Functions
RvSipTransportBadSyntaxMsgEv()

**RvSipTransportBadSyntaxMsgEv()**

**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**

```c
typedef RvStatus
(RVCALLCONV * RvSipTransportBadSyntaxMsgEv)(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipAppTransportMgrHandle hAppTransportMgr,
    IN RvSipMsgHandle hMsgReceived,
    OUT RvSipTransportBsAction *peAction);
```

**PARAMETERS**

- **hTransportMgr**
  The handle to the `TransportMgr`.

- **hAppTransportMgr**
  The application handle. You supply this handle when setting the event handles.

- **hMsgReceived**
  The received bad-syntax message.

- **peAction**
  The decision of the user on how the SIP Stack should handle this message.
**Transport Callback Functions**

RvSipTransportBadSyntaxMsgEv()

**RETURN VALUES**

Returns RvStatus. (The returned status is ignored in the current SIP Stack version.)
Transport Callback Functions
RvSipTransportBadSyntaxStartLineMsgEv()

RvSipTransportBadSyntaxStartLineMsgEv()

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 * RvSipTransportBadSyntaxStartLineMsgEv)(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipAppTransportMgrHandle hAppTransportMgr,
    IN RvSipMsgHandle hMsgReceived,
    OUT RvSipTransportBsAction *peAction);

PARAMETERS

hTransportMg
The 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.)
Transport Callback Functions
RvSipTransportConnectionCreatedEv()

**RvSipTransportConnectionCreatedEv()**

**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 the sending and receiving of data.

**SYNTAX**
```c
typedef void
(RVCALLCONV * RvSipTransportConnectionCreatedEv)(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipAppTransportMgrHandle hAppTransportMgr,
    IN RvSipTransportConnectionHandle hConn,
    OUT RvSipTransportConnectionAppHandle *phAppConn,
    OUT RvBool *pbDrop);
```

**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 created *connection*.
- **phAppConn**
The handle that the application set for the *connection*.
Transport Callback Functions
RvSipTransportConnectionCreatedEv()

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.
RvSipTransportBufferReceivedEv()

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 * RvSipTransportBufferReceivedEv)(
    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.
Transport Callback Functions
RvSipTransportBufferReceivedEv()

**pSenderAddrDetails**
A pointer to RvSipTransportAddr, which contains details of the address from which the message was sent.

**hConn**
The handle to 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.
Transport Callback Functions
RvSipTransportBufferToSendEv()

RvSipTransportBufferToSendEv()

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 or not the Transport layer should transmit the message to its destination. Note that discarding a message does not affect the state of the sender object. Discarding a message can simulate message loss on the net.

SYNTAX

typedef void (RVCALLCONV * RvSipTransportBufferToSendEv)(
    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
The 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.
Transport Callback Functions
RvSipTransportBufferToSendEv()

**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 to 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.
Transport Callback Functions
RvSipTransportConnectionParserResultEv()

**RvSipTransportConnectionParserResultEv()**

**DESCRIPTION**
Indicates 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 * RvSipTransportConnectionParserResultEv)(
    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 has arrived.

- **hAppConn**
  The handle that the application sets for the `connection`. 
Transport Callback Functions
RvSipTransportConnectionParserResultEv()

**LegalSyntax**
RV_TRUE if the parser did not discover bad syntax. Otherwise, RV_FALSE.

**RETURN VALUES**
None.
RvSipTransportMsgToSendEv()

**DESCRIPTION**
Notifies the application that a new message is about to be sent. The application can decide whether or not the Transport layer should transmit the message to its destination.

**SYNTAX**
```c
typedef RvBool (RVCALLCONV * RvSipTransportMsgToSendEv)(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipAppTransportMgrHandle hAppTransportMgr,
    IN RvChar *msgBuffer,
    IN RvUint bufferLen);
```

**PARAMETERS**
- **hTransportMgr**
  A handle to the `TransportMgr`.
- **hAppTransportMgr**
  The application handle. You supply this handle when setting the event handles.
- **msgBuffer**
  The message that is about to be sent (given in a consecutive buffer).
- **bufferLen**
  The length of the message buffer.

**RETURN VALUES**
Returns RV_TURE to transmit the message. Returns RV_FALSE to discard the message without transmitting it to the destination.
RvSipTransportMsgReceivedEv()

DESCRIPTION
Notifies the application that a new message was received. The application can decide whether or not the Transport layer should discard the message. If the application decides to discard the message, SIP Stack will not process the message.

SYNTAX

typedef RvBool (RVCALLCONV * RvSipTransportMsgReceivedEv)(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipAppTransportMgrHandle hAppTransportMgr,
    IN RvSipMsgHandle hMsgReceived);

PARAMETERS

hTransportMgr
A handle to the TransportMgr.

hAppTransportMgr
The application handle. You supply this handle when setting the event handles.

hMsgReceived
The received message.

RETURN VALUES
Returns RV_TRUE to accept the message and process it in the SIP Stack.
Returns RV_FALSE to discard the message without further processing.
RvSipTransportConnectionTlsStatusEv()

DESCRIPTION
Notifies the application about the status of the TLS connection. The application can stop the establishment of the connection with the pbContinue parameter.

SYNTAX
typedef RvStatus (*RvSipTransportConnectionTlsStatusEv)(
    IN RvSipTransportConnectionHandle hConn,
    IN RvSipTransportConnectionAppHandle hAppConnection,
    IN RvSipTransportConnectionTlsStatus eStatus,
    OUT RvBool* pbContinue);

PARAMETERS

hConn
The connection handle.

hAppConnection
The application handle to the connection.

eStatus
The connection status.

pbContinue
RV_TRUE if you wish to complete the connection establishment. Otherwise, RV_FALSE.

RETURN VALUES
Returns RvStatus.
RvSipTransportMsgReceivedExtEv()

DESCRIPTION

Notifies the application that a new message was received. The application can decide whether or not the Transport layer should discard the message, in which case the SIP Stack will not process this message.

SYNTAX

typedef RvStatus (*RvSipTransportMsgReceivedExtEv)(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipAppTransportMgrHandle hAppTransportMgr,
    IN RvSipMsgHandle hMsgReceived,
    IN RvSipTransportRcvdMsgDetails *pRcvdMsgDetails,
    OUT RvBool *pbProcessMsg);

PARAMETERS

hTransportMgr
A handle to the TransportMgr.

hAppTransportMgr
The application handle. You supply this handle when setting the event handles.

hMsgReceived
The handle to the received message.

pRcvdMsgDetails
A pointer to the RvSipTransportRcvdMsgDetails structure, containing information about the received message.

pbProcessMsg
RV_TRUE if you wish the SIP Stack to processes the incoming message. RV_FALSE if you wish the SIP Stack to discard the message.
Transport Callback Functions
RvSipTransportMsgReceivedExtEv()

**RETURN VALUES**

None.
RvSipTransportConnectionDataReceivedEv()

DESCRIPTION

Notifies the application that data which forms the message was read from the socket of the incoming connection.

SYNTAX

typedef RvStatus (*RvSipTransportConnectionDataReceivedEv)(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipAppTransportMgrHandle hAppTransportMgr,
    IN RvSipTransportConnectionHandle hConn,
    IN RvSipTransportConnectionAppHandle hAppConn,
    IN RvChar *buff,
    IN RvInt32 buffSize,
    OUT RvBool *pbDiscard);

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 on which the message data was received.

hAppConn
The handle that the connection sets into the application.

buff
The buffer containing the read data.

buffSize
The size of the read data in bytes.
Transport Callback Functions
RvSipTransportConnectionDataReceivedEv()

pbDiscard
If RV_TRUE, the received data will be discarded.

RETURN VALUES
None.
**RvSipTransportConnectionServerReuseEv()**

**DESCRIPTION**

This callback function is called when an incoming request has an alias parameter in its top-most Via header. This callback informs the 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**

```c
typedef RvStatus (*RvSipTransportConnectionServerReuseEv)(
    IN RvSipTransportMgrHandle hTransportMgr,
    IN RvSipAppTransportMgrHandle hAppTransportMgr,
    IN RvSipTransportConnectionHandle hConn,
    IN RvSipTransportConnectionAppHandle hAppConn);
```

**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 set for the `connection`.

**RETURN VALUES**

None.
Transport Object Event Handler
RvSipTransportConnectionServerReuseEv()

The event handler is:

- RvSipTransportObjectEventHandler()
RvSipTransportObjectEventHandler()

**DESCRIPTION**
Processes the event of a specific object.

**SYNTAX**

typedef RvStatus (RVCALLCONV* 
RvSipTransportObjectEventHandler)(
    IN void *pObj,
    IN RvInt32 eventReason);

**PARAMETERS**

- **pObj**
  A pointer to the object to terminate.

- **eventReason**
  The reason for the event.

**RETURN VALUES**

Returns RvStatus.
Transport Object Event Handler
RVsipTransportObjectEventHandler()
RESOLVER API TYPE DEFINITIONS

This section includes the SIP Stack Resolver type definitions and callback functions defined in the RvSipResolverTypes.h header file.

This section includes:
- Handle Type Definitions
- Resolver Type Definitions
- Resolver Callback Functions
Handle Type Definitions

The handle type definitions are:

- RvSipResolverMgrHandle
- RvSipResolverHandle
- RvSipAppResolverHandle
RvSipResolverMgrHandle

DESCRIPTION
The declaration of a handle to the ResolverMgr.

SYNTAX
RV_DECLARE_HANDLE(RvSipResolverMgrHandle);
RvSipResolverHandle

DESCRIPTION
The declaration of a handle to the resolver.

SYNTAX
RV_DECLARE_HANDLE(RvSipResolverHandle);
RvSipAppResolverHandle

DESCRIPTION
The declaration of a handle to the application resolver.

SYNTAX
RV_DECLARE_HANDLE(RvSipAppResolverHandle);
Resolver Type Definitions

The Resolver type definitions are:

- `RvSipResolverMode`
- `RvSipResolverScheme`
RvSipResolverMode

DESCRIPTION
Denotes the mode by which the resolver will act.

SYNTAX

typedef enum {
    RVSIP_RESOLVER_MODE_UNDEFINED = -1,
    RVSIP_RESOLVER_MODE_FIND_TRANSPORT_BY_NAPTR,
    RVSIP_RESOLVER_MODE_FIND_TRANSPORT_BY_3WAY_SRV,
    RVSIP_RESOLVER_MODE_FIND_HOSTPORT_BY_SRV_STRING,
    RVSIP_RESOLVER_MODE_FIND_HOSTPORT_BY_TRANSPORT,
    RVSIP_RESOLVER_MODE_FIND_IP_BY_HOST,
    RVSIP_RESOLVER_MODE_FIND_URI_BY_NAPTR
} RvSipResolverMode;

PARAMETERS

RVSIP_RESOLVER_MODE_UNDEFINED
No resolution mode.

RVSIP_RESOLVER_MODE_FIND_TRANSPORT_BY_NAPTR
Instructs the resolver to try obtaining NAPTR records for a domain. The resolver will use the protocol field of the NAPTR record to determine the transport. (For example, when getting the available transports for “hp.com” the result will be a list of SRV record pointers.)

RVSIP_RESOLVER_MODE_FIND_TRANSPORT_BY_3WAY_SRV
Instructs the resolver to try finding a transport for a domain by applying SRV queries of _service._protocol.domain, until the resolver receives a successful answer. (For example, trying to find hp.com will send queries such as _scheme._protocol.hp.com on UDP, TCL and finally TLS. If the resolver gets a positive answer, it will stop querying, and as a result, provide a host pointer and a port number.)
Resolver Type Definitions
RvSipResolverMode

RVSIP_RESOLVER_MODE_FIND_HOSTPORT_BY_SRV_STRING
Instructs the resolver to send one SRV query with the string supplied and to save the host and port retrieved in the answer in the DNS list (for example, trying to get the host and port for _sip._udp.hp.com).

RVSIP_RESOLVER_MODE_FIND_HOSTPORT_BY_TRANSPORT
Instructs the resolver to create one service._protocol.domain string and to try to send an SRV query for it. The host and port retrieved in the answer will be stored in the DNS list. (For example, trying to get the TCP/pres SRV record for “hp.com” will send a _pres._udp.hp.com SRV query and will result in a host pointer and a port number.)

RVSIP_RESOLVER_MODE_FIND_IP_BY_HOST
Instructs the resolver to try finding the IP of a specific host. The resolver will try both IPv4 and IPv6 (using A and AAAA queries respectively). (For example, when trying to get the IP for host1.hp.com, the result will be an IP address.)

RVSIP_RESOLVER_MODE_FIND_URI_BY_NAPTR
Gets the NAPTR record of an ENUM record in a DNS server. (For example, trying to resolve a phone number such as +97237679623 will result in a NAPTR query—for 3.2.6.9.7.6.7.3.2.7.9.e164.arpa. the result will be a regular expression.)
RvSipResolverScheme

DESCRIPTION
Enumerates the different schemes that the resolver can use for SRV queries. The eScheme parameter specifies the scheme to concatenate to the query string parameter when an SRV query is applied to retrieve the transport.

SYNTAX
typedef enum {
    RVSIP_RESOLVER_SCHEME_UNDEFINED = -1,
    RVSIP_RESOLVER_SCHEME_SIP,
    RVSIP_RESOLVER_SCHEME_PRES,
    RVSIP_RESOLVER_SCHEME_IM
} RvSipResolverScheme;

PARAMETERS

RVSIP_RESOLVER_SCHEME_UNDEFINED
No scheme.

RVSIP_RESOLVER_SCHEME_SIP
The “sip”/”sips” scheme.

RVSIP_RESOLVER_SCHEME_PRES
The “pres” scheme.
Resolver Callback Functions

The Resolver callback function is:
- `RvSipResolverReportData()`
RvSipResolverReportData()

DESCRIPTION
Using this callback, the resolver reports the results of a DNS algorithm requested by the application. The application has to supply the callback function pointer when calling the RvSipResolverResolve() API function. This callback will be called whenever the algorithm block has ended, and will indicate if the algorithm block was successful or not. When this callback is called and indicates that there is no error, the DNS list will contain the required answers. You can obtain the list by calling the RvSipResolverGetDnsList() API function.

SYNTAX
typedef RvStatus (RVCALLCONV *RvSipResolverReportDataEv) (  
    IN RvSipResolverHandle hResolver,  
    IN RvSipAppResolverHandle hAppRslv,  
    IN RvBool bError,  
    IN RvSipResolverMode eMode);

PARAMETERS
hResolver
The resolver handle.

hAppRslv
The application handle to the resolver.

bError
Indicates whether or not the resolution was successful.

eMode
The resolver mode that was used.

RETURN VALUES
Ignored.
Resolver Callback Functions
RvSipResolverReportData()
This section includes The SIP Stack RPOOL type definitions defined in the rpool_API.h header file.

This section includes:

- Handle Type Definitions
- RPOOL Type Definitions
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);

PARAMETERS
None.

RETURN VALUES
None.
**Handle Type Definitions**

**HPAGE**

---

**HPAGE**

**DESCRIPTION**

Defines the handle to a page inside a memory pool.

**SYNTAX**

```c
typedef RvInt32 HPAGE;
```
NULL_PAGE

DESCRIPTION
Defines an invalid page.

SYNTAX
#define NULL_PAGE (NULL)
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**

```c
typedef struct{
    HRPOOL hPool;
    HPAGE hPage;
    RvInt32 offset;
}RPOOL_Ptr;
```
RPOOL Type Definitions
RPOOL_Ptr
This section provides headers for the Mid-layer of the SIP Stack found in the RvSipMidTypes.h header file. The Mid-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
The type definitions are:

- \texttt{RvSipMidMgrHandle}
- \texttt{RvSipMidTimerHandle}
- \texttt{RvSipMidCfg}
- \texttt{RvSipMidSelectEvent}
- \texttt{RvSipMidSelectEv()}
- \texttt{RvSipMidTimerExpEv()}
RvSipMidMgrHandle

DESCRIPTION
Declares a handle to the Mid-LayerMgr.

SYNTAX
RV_DECLARE_HANDLE(RvSipMidMgrHandle);
Type Definitions
RvSipMidTimerHandle

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**
```c
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.
Type Definitions
RvSipMidTimerExpEv()

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.
This section includes the Common type definitions that are used by the different SIP Stack layers. These type definitions are defined in the RvSipCommonTypes.h and RvSipCommonList.h header files.

This section includes:

- Common Type Definitions
- Common List Type Definitions
Common Type Definitions

COMMON TYPE DEFINITIONS

The Common type definitions are:

- RvSipCommonStackObjectType
- RvSipListLocation
- RvSipLogFilters
- RvSipTimers
RvSipCommonStackObjectType

DESCRIPTION
The different types of objects available in the SIP Stack.

SYNTAX

typedef enum{
    RVSIP_COMMON_STACK_OBJECT_TYPE_UNDEFINED = -1,
    RVSIP_COMMON_STACK_OBJECT_TYPE_CALL_LEG,
    RVSIP_COMMON_STACK_OBJECT_TYPE_TRANSACTION,
    RVSIP_COMMON_STACK_OBJECT_TYPE_CALL_TRANSACTION,
    RVSIP_COMMON_STACK_OBJECT_TYPE_SUBSCRIPTION,
    RVSIP_COMMON_STACK_OBJECT_TYPE_NOTIFICATION,
    RVSIP_COMMON_STACK_OBJECT_TYPE_REG_CLIENT,
    RVSIP_COMMON_STACK_OBJECT_TYPE_CONNECTION,
    RVSIP_COMMON_STACK_OBJECT_TYPE_APP_OBJECT,
    RVSIP_COMMON_STACK_OBJECT_TYPE_CALL_INVITE,
    RVSIP_COMMON_STACK_OBJECT_TYPE_TRANSMITTER,
}RvSipCommonStackObjectType;
RvSipListLocation

DESCRIPTION

Represents the location of an element in a list. This enumeration is used in various API functions that are used to go over a list kept in the SIP Stack.

SYNTAX

typedef enum{
    RVSIP_FIRST_ELEMENT = 0,
    RVSIP_LAST_ELEMENT,
    RVSIP_NEXT_ELEMENT,
    RVSIP_PREV_ELEMENT
}RvSipListLocation;
RvSipLogFilters

**DESCRIPTION**
Defines the different log filters that can be combined to control how the SIP Stack produces log messages.

**SYNTAX**
```c
typedef enum{
    RVSIP_LOG_DEBUG_FILTER = 0x01,
    RVSIP_LOG_INFO_FILTER = 0x02,
    RVSIP_LOG_WARN_FILTER = 0x04,
    RVSIP_LOG_ERROR_FILTER = 0x08,
    RVSIP_LOG_EXCEP_FILTER = 0x10,
    RVSIP_LOG_LOCKDBG_FILTER = 0x20,
    RVSIP_LOG_ENTER_FILTER = 0x40, /*only for core logging*/
    RVSIP_LOG_LEAVE_FILTER = 0x80 /*only for core logging*/
}RvSipLogFilters;
```
RvSipTimers

DESCRIPTION
Sets different timeout values to different SIP Stack objects, overriding the values received from the SIP Stack configuration. This structure is also used to control the number of transmissions performed for requests and responses.

SYNTAX

typedef struct{
    /*timeouts*/
    RvInt32 T1Timeout;
    RvInt32 T2Timeout;
    RvInt32 T4Timeout;
    RvInt32 genLingerTimeout;
    RvInt32 inviteLingerTimeout;
    RvInt32 cancelInviteNoResponseTimeout;
    RvInt32 generalRequestTimeoutTimeout;
    RvInt32 provisionalTimeout;

    /*retransmissions limitation*/
    RvInt8 maxInviteRequestRetransmissions;
    RvInt8 maxInviteResponseRetransmissions;
    RvInt8 maxGeneralRequestRetransmissions;
}
}RvSipTimers;

PARAMETERS

T1Timeout

Determines several timers as defined in RFC 3261. For example, on UDP, a UAC retransmits INVITE request at an interval that starts with T1 seconds, and doubles after every retransmission. On UDP, a UAC retransmits a general request at an interval that starts with T1 seconds, and doubles until it reaches T2.
Common Type Definitions

RvSipTimers

**T2Timeout**

Determines the maximum retransmission interval. The value of this parameter cannot be less than 4000. For example, on UDP, general requests are retransmitted at an interval which starts at T1 and doubles until it reaches T2. If a provisional response is received, retransmissions continue but at an interval of T2.

**T4Timeout**

Represents the amount of time the network takes to clear messages between client and server *transactions*. For example, on UDP, T4 determines the time that a UAS waits after receiving an ACK message and before terminating the *transaction*.

**genLingerTimeout**

After a server sends a final response, the server cannot be sure that the client has received the response message. The server should be able to retransmit the response upon receiving retransmissions of the request for genLingerTimer milliseconds.

**Remarks:** If no value was given - genLingerTimeout = 64*T1

**inviteLingerTimeout**

After sending an ACK message for an INVITE final response, a client cannot be sure that the server has received the ACK message. The client should be able to retransmit the ACK upon receiving retransmissions of the final response for inviteLingerTimeout milliseconds.

**cancelInviteNoResponseTimer**

After sending a CANCEL for an INVITE request, a client will wait for a final response until the cancelInviteNoResponseTimer expires.

**Remarks:** If no value was given, cancelInviteNoResponseTimer = 64*T1. The value must be greater than 0.

**generalRequestTimeout**

After sending a general request, the *transaction* enters the CLIENT_GENERAL_REQUEST_SENT state. If no response is received when generalRequestTimeoutTimer is expired, the *transaction* is terminated automatically (in any transport).
Remarks: If no value is given, generalRequestTimeoutTimer = 64*T1

provisionalTimeout
The provisionalTimer is set when a provisional response is received for an INVITE request. The transaction will stop retransmissions of the INVITE request and wait for a final response until the provisionalTimeout expires. If you set the provisionalTimeout to zero, no timer is set. The INVITE transaction will wait indefinitely for the final response.

maxInviteRequestRetransmissions
The limitation for the number of INVITE request transmissions.

maxInviteResponseRetransmissions
The limitation for the number of INVITE response transmissions.

maxGeneralRequestRetransmissions
The limitation for the number of general request transmissions.
The Common List type definitions are:

- RvSipCommonListHandle
- RvSipCommonListElemHandle
Common List Type Definitions
RvSipCommonListHandle

RvSipCommonListHandle

DESCRIPTION
The handle to a sip-common-list object (sip-common-list).

SYNTAX
RV_DECLARE_HANDLE(RvSipCommonListHandle);
RvSipCommonListElemHandle

The handle to a sip-common-list element (sip-common-list element) object.

**SYNTAX**

RV_DECLARE_HANDLE(RvSipCommonListElemHandle);
Common List Type Definitions
RvSipCommonListElemHandle
The structures in this section define the resource status for each of the SIP Stack modules. You should supply these structures when using the `RvSipStackGetResources()` function. This section includes the SIP Stack Resources type definitions defined in the `RvSipResourcesTypes.h` header file. The type definitions included in this section are:

- Resource Type Definitions
Resource Type Definitions

The Resource type definitions are:

- RvSipResource
- RvSipStackResources
- RvSipCallLegResources
- RvSipRegClientResources
- RvSipTranscResources
- RvSipTransportResources
- RvSipSubsResources
- RvSipTransmitterResources
- RvSipResolverResources
RvSipResource

DESCRIPTION
Defines a general resource structure for the SIP Stack modules.

SYNTAX

typedef struct {
    RvUint32 numOfAllocatedElements;
    RvUint32 currNumOfUsedElements;
    RvUint32 maxUsageOfElements;
}RvSipResource;

PARAMETERS

numOfAllocatedElements
The total number of elements that the SIP Stack allocated.

currNumOfUsedElements
The number of elements that are currently being used.

maxUsageOfElements
The maximum number of elements that were used concurrently until the present.
Resource Type Definitions
RvSipStackResources

RvSipStackResources

DESCRIPTION
Defines the resource structure of the Stack Manager module.

SYNTAX
typedef struct{
    RvSipResource msgPoolElements;
    RvSipResource generalPoolElements;
    RvSipResource headerPoolElements;
    RvSipResource timerPool;
}RvSipStackResources;
RvSipCallLegResources

DESCRIPTION
Defines the resource structure of the Call-leg module.

SYNTAX

typedef struct{
    RvSipResource calls;
    RvSipResource transcLists;
    RvSipResource transcHandles;
    RvSipResource inviteLists;
    RvSipResource inviteObjects;
}RvSipCallLegResources;
Resource Type Definitions
RvSipRegClientResources

RvSipRegClientResources

DESCRIPTION
Defines the resource structure of the Register-Client module.

SYNTAX
typedef struct{
    RvSipResource    regClients;
}RvSipRegClientResources;
RvSipTranscResources

DESCRIPTION
Defines the resource structure of the Transaction module.

SYNTAX

typedef struct{
    RvSipResource transactions;
}RvSipTranscResources;
**RvSipTransportResources**

**DESCRIPTION**

Defines the resources of the Transport module.

**SYNTAX**

typedef struct{
    RvSipResource connections;
    RvSipResource pQueueElements;
    RvSipResource pQueueEvents;
    RvSipResource readBuffers;
    RvSipResource connHash;
    RvSipResource ownersHash;
    RvSipResource tlsSessions;
    RvSipResource tlsEngines;
    RvSipResource oorEvents; /*oor = out of resources*/
}RvSipTransportResources;
**RvSipSubsResources**

**DESCRIPTION**

Defines the resource structure of the Subscription module.

**SYNTAX**

```c
typedef struct{
    RvSipResource subscriptions;
    RvSipResource notifications;
    RvSipResource notifyLists;
} RvSipSubsResources;
```
Resource Type Definitions
RvSipTransmitterResources

RvSipTransmitterResources

DESCRIPTION
Defines the resource structure of the Transmitter module.

SYNTAX
typedef struct{
  RvSipResource transmitters;
}RvSipTransmitterResources;
RvSipResolverResources

**DEFINITION**
Defines the resource structure of the Resolver module.

**SYNTAX**
```c
typedef struct{
    RvSipResource resolvers;
}RvSipResolverResources;
```
Resource Type Definitions
RvSipResolverResources
The status codes in Table 26-1 include SIP specific codes defined in the `RV_SIP_DEF.h` header file and the general status codes defined in the `rverrort.h` header file. The status codes may be returned by SIP Stack API functions as RvStatus.

Table 26-1  Status Codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RV_OK</td>
<td>The function was completed successfully.</td>
</tr>
<tr>
<td>RV_ERROR_UNKNOWN</td>
<td>Failure of unspecified type.</td>
</tr>
<tr>
<td>RV_ERROR_OUTOFRESOURCES</td>
<td>The function can not be executed due to limited resources.</td>
</tr>
<tr>
<td>RV_ERROR_BADPARAM</td>
<td>A parameter passed to a function is illegal.</td>
</tr>
<tr>
<td>RV_ERROR_NULLPTR</td>
<td>The required pointer parameter was a NULL pointer.</td>
</tr>
<tr>
<td>RV_ERROR_OUTOF RANGE</td>
<td>A parameter that was passed to a function is out of range.</td>
</tr>
<tr>
<td>RV_ERROR_DESTRUCTED</td>
<td>The referred object was already terminated.</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RV_ERROR_NOT_SUPPORTED</td>
<td>The request is not supported under the current configuration.</td>
</tr>
<tr>
<td>RV_ERROR_UNINITIALIZED</td>
<td>The object is uninitialized.</td>
</tr>
<tr>
<td>RV_ERROR_TRY_AGAIN</td>
<td>The action cannot be completed—try again later.</td>
</tr>
<tr>
<td>RV_ERROR_ILLEGAL_ACTION</td>
<td>The requested action is illegal—usually an illegal action occurring in the current state.</td>
</tr>
<tr>
<td>RV_ERROR_NETWORK_PROBLEM</td>
<td>Action failed due to network problems.</td>
</tr>
<tr>
<td>RV_ERROR_INVALID_HANDLE</td>
<td>A handle passed to a function is illegal.</td>
</tr>
<tr>
<td>RV_ERROR_NOT_FOUND</td>
<td>The requested item cannot be found.</td>
</tr>
<tr>
<td>RV_ERROR_INSUFFICIENT_BUFFER</td>
<td>The buffer is too small.</td>
</tr>
<tr>
<td>RV_ERROR_ILLEGAL_SYNTAX</td>
<td>The parser identified a syntax error, or a parser error occurred.</td>
</tr>
<tr>
<td>RV_ERROR_OBJECT_ALREADY_EXISTS</td>
<td>An object with this unique set of values already exists.</td>
</tr>
<tr>
<td>RV_ERROR_NUM_OF_THREADS_DECREASED</td>
<td>The processing thread exited abnormally.</td>
</tr>
</tbody>
</table>
INDEX

H
HPAGE 1026
HRPOOL 1025

N
NULL_PAGE 1027

R
RPOOL_AppendFromExternalToPage() 631
RPOOL_Construct() 625
RPOOL_CopyToExternal() 630
RPOOL_Destruct() 627
RPOOL_FreePage() 629
RPOOL_GetPage() 628
RPOOL_GetResources() 633
RPOOL_Ptr 1029
RPOOL_Strlen() 632
RV_Status 1063
RvSipAppAuthenticatorHandle 858
RvSipAppCallLegHandle 687
RvSipAppCallLegInviteHandle 689
RvSipAppNotifyHandle 881
RvSipAppRegClientHandle 837
RvSipAppResolverHandle 1015
RvSipAppStackHandle 754
RvSipAppSubhHandle 879
RvSipAppTranscHandle 776
RvSipAppTransmiterHandle 813
RvSipAppTransportMgrHandle 936
RvSipAuthenticatorAddProxyAuthorizationToMsg() 384
RvSipAuthenticatorAuthorizationReadyEv() 869
RvSipAuthenticatorCredentialsSupported() 379
RvSipAuthenticatorEvHandlers 861
RvSipAuthenticatorGetSharedSecretEv() 865
RvSipAuthenticatorGetStackInstance() 393
RvSipAuthenticatorHandle 857
RvSipAuthenticatorMD5EntityBodyEv() 867
RvSipAuthenticatorMD5Ev() 864
RvSipAuthenticatorNonceCountUsageEv() 873
RvSipAuthenticatorPrepareAuthenticationInfoHeader() 388
RvSipAuthenticatorPrepareAuthorizationHeader() 386
RvSipAuthenticatorSetAppHandle() 392
RvSipAuthenticatorSetEvHandlers() 391
RvSipAuthenticatorSetNonceCount() 396
RvSipAuthenticatorSetProxyAuthInfo() 394
RvSipAuthenticatorUnsupportedChallengeEv() 871
RvSipAuthenticatorVerifyCredentials() 380
RvSipAuthenticatorVerifyCredentialsExt() 382
RvSipAuthObjGetAppContext() 400
RvSipAuthObjGetAuthenticationHeader() 397
RvSipAuthObjHandle 859
RvSipAuthObjSetAppContext() 399
RvSipAuthObjSetUserInfo() 398
RvSipCallLegAccept() 45
RvSipCallLegAck() 60
RvSipCallLegAuthBegin() 120
RvSipCallLegReInviteCreate()  64
RvSipCallLegReInviteCreatedEv()  722
RvSipCallLegReInvitecurrentState()  69
RvSipCallLegReInviteRequest()  66
RvSipCallLegReInviteSetAppHandle()  65
RvSipCallLegReInviteStateChangedEv()  723
RvSipCallLegReInviteTerminate()  68
RvSipCallLogReject()  47
RvSipCallLegReplacesCompareReplacesToCallLeg()  132
RvSipCallLegReplacesGetMatchedCallExt()  131
RvSipCallLegReplacesReason  706
RvSipCallLegReplacesStatus  705
RvSipCallLegResetOutboundMsg()  100
RvSipCallLegResources  1055
RvSipCallLegRespondUnauthenticated()  123
RvSipCallLogRespondUnauthenticatedDigest()  125
RvSipCallLegRestoreConnectedCall()  117
RvSipCallLegRestoreOldVersionConnectedCall()  118
RvSipCallLegSendPrack()  62
RvSipCallLegSendPrackResponse()  63
RvSipCallLegSessionTimerGetAlertTime()  145
RvSipCallLegSessionTimerGetMinSEValue()  148
RvSipCallLegSessionTimerGetNegotiationParams()  141
RvSipCallLegSessionTimerGetRefresherType()  147
RvSipCallLegSessionTimerGetSessionExpiresValue()  149
RvSipCallLegSessionTimerInviteRefresh()  134
RvSipCallLegSessionTimerNegotiationFaultEvent()  731
RvSipCallLegSessionTimerNegotiationReason  709
RvSipCallLegSessionTimerNotificationEv()  730
RvSipCallLegSessionTimerNotificationReason  708
RvSipCallLegSessionTimerRefreshAlertEv()  729
RvSipCallLegSessionTimerRefresherPreference()  710
RvSipCallLegSessionTimerRefresherPreference()  710
RvSipCallLegSessionTimerSetAlertTime()  146
RvSipCallLegSessionTimerSetPreferenceParameters()  137
RvSipCallLegSetAddAuthInfoToMsgFlag()  112
RvSipCallLegSetAppHandle()  88
RvSipCallLegSetCallId()  74
RvSipCallLegSetConnection()  105
RvSipCallLegSetCSeq()  85
RvSipCallLegSetForceOutboundAddrFlag()  111
RvSipCallLegSetForked1xxTimerTimeout()  172
RvSipCallLegSetForkingEnabledFlag()  170
RvSipCallLegSetFromHeader()  76
RvSipCallLegSetLocalAddress()  160
RvSipCallLegSetLocalContactAddress()  82
RvSipCallLegSetOutboundDetails()  164
RvSipCallLegSetOutboundMsg()  101
RvSipCallLegSetPersistency()  103
RvSipCallLegSetRejectStatusCodesOnCreation()  108
RvSipCallLegSetRemoteContactAddress()  80
RvSipCallLegSetReplacesHeader()  97
RvSipCallLegSetToHeader()  78
RvSipCallLegSetTranscTimers()  113
RvSipCallLegState  691
RvSipCallLegStateChangedEv()  719
RvSipCallLegStateChangeReason  692
RvSipCallLegStoreConnectedCall()  116
RvSipCallLegTerminate()  58
RvSipCallLegTranscCreate()  53
RvSipCallLegTranscCreatedEv()  737
RvSipCallLegTranscRequest()  54
RvSipCallLegTranscResponse() 56
RvSipCallLegTranscSessionTimerGeneralRefresh() 135
RvSipCallLegTranscSessionTimerGetNegotiationParams() 143
RvSipCallLegTranscSessionTimerSetPreferenceParams() 139
RvSipCallLegTranscState 702
RvSipCallLegTranscStateChangedEv() 739
RvSipCallLegTranscTerminate() 57
RvSipCallLegUseFirstRouteForInitialRequest() 110
RvSipCommonListConstruct() 669
RvSipCommonListConstructOnPage() 670
RvSipCommonListDestruct() 671
RvSipCommonListElemHandle 1049
RvSipCommonListGetElem() 676
RvSipCommonListGetPage() 680
RvSipCommonListGetPool() 679
RvSipCommonListHandle 1048
RvSipCommonListsEmpty() 678
RvSipCommonListPushElem() 672
RvSipCommonListRemoveElem() 674
RvSipCommonStackObjectType 1041
RvSipListLocation 1042
RvSipLogFilters 1043
RvSipMidCfg 1035
RvSipMidConstruct() 641
RvSipMidDecodeB64() 662
RvSipMidDestruct() 643
RvSipMidEncodeB64() 661
RvSipMidEnd() 640
RvSipMidInit() 639
RvSipMidMemAlloc() 645
RvSipMidMemFree() 646
RvSipMidMgrHandle 1033
RvSipMidPollEventsHandling() 660
RvSipMidPollGetEventsRegistration() 658
RvSipMidPrepareDestruct() 642
RvSipMidSelectCallOn() 651
RvSipMidSelectEv() 1037
RvSipMidSelectEvent 1036
RvSipMidSelectEventsHandling() 656
RvSipMidSelectGetEventsRegistration() 654
RvSipMidSelectGetMaxDesc() 653
RvSipMidSelectGetMaxDescs() 652
RvSipMidSetLog() 644
RvSipMidTimeInMillis() 647
RvSipMidTimeInSeconds() 648
RvSipMidTimerExpEv() 1038
RvSipMidTimerHandle 1034
RvSipMidTimerReset() 650
RvSipMidTimerSet() 649
RvSipMidTlsLockingCallback() 663
RvSipMidTlsSelectThreadIdCallback() 664
RvSipNotifyAccept() 464
RvSipNotifyDetachOwner() 467
RvSipNotifyGetOutboundMsg() 468
RvSipNotifyGetStackInstance() 469
RvSipNotifyGetSubsState() 470
RvSipNotifyHandle 880
RvSipNotifyReject() 465
RvSipNotifySend() 463
RvSipNotifySetReferNotifyBody() 494
RvSipNotifySetSubscriptionStateParams() 471
RvSipNotifyTerminate() 466
RvSipRegClientAuthenticate() 370
RvSipRegClientAuthObjGet() 372
RvSipRegClientAuthObjRemove() 373
RvSipRegClientDetachFromMgr() 325
RvSipRegClientDetachOwner() 323
RvSipRegClientDNSContinue() 359
RvSipRegClientDNSGetList() 361
RvSipRegClientDNSGiveUp() 358
RvSipRegClientDNSResolveEndRequest() 360
RvSipRegClientEvHandlers 843
RvSipRegClientFinalDestResolvedEv() 851
RvSipRegClientGetAppHandle() 343
RvSipRegClientGetCallId() 341
RvSipRegClientGetConnection() 353
RvSipRegClientGetCSeqStep() 346
RvSipRegClientGetCurrentState() 344
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RvSipStackGetRegClientMgrHandle()</td>
<td>18</td>
</tr>
<tr>
<td>RvSipStackGetResolverMgrHandle()</td>
<td>23</td>
</tr>
<tr>
<td>RvSipStackGetResources()</td>
<td>27</td>
</tr>
<tr>
<td>RvSipStackGetStatistics()</td>
<td>28</td>
</tr>
<tr>
<td>RvSipStackGetSubsMgrHandle()</td>
<td>21</td>
</tr>
<tr>
<td>RvSipStackGetTransactionMgrHandle()</td>
<td>16</td>
</tr>
<tr>
<td>RvSipStackGetTransmitterMgrHandle()</td>
<td>22</td>
</tr>
<tr>
<td>RvSipStackGetTransportMgrHandle()</td>
<td>20</td>
</tr>
<tr>
<td>RvSipStackGetVersion()</td>
<td>12</td>
</tr>
<tr>
<td>RvSipStackHandle</td>
<td>753</td>
</tr>
<tr>
<td>RvSipStackInitCfg()</td>
<td>5</td>
</tr>
<tr>
<td>RvSipStackIsLogFilterExist()</td>
<td>26</td>
</tr>
<tr>
<td>RvSipStackIsReplacesSupported()</td>
<td>32</td>
</tr>
<tr>
<td>RvSipStackIsSessionTimerSupported()</td>
<td>31</td>
</tr>
<tr>
<td>RvSipStackMgrIsEnhancedDnsFeatureEnabled()</td>
<td>29</td>
</tr>
<tr>
<td>RvSipStackMgrIsTlsFeatureEnabled()</td>
<td></td>
</tr>
<tr>
<td>RvSipStackModule</td>
<td>756</td>
</tr>
<tr>
<td>RvSipStackPrintLogEntryEv()</td>
<td>769</td>
</tr>
<tr>
<td>RvSipStackPrintLogEntryEv()</td>
<td>769</td>
</tr>
<tr>
<td>RvSipStackProcessEvents()</td>
<td>7</td>
</tr>
<tr>
<td>RvSipStackResources</td>
<td>1054</td>
</tr>
<tr>
<td>RvSipStackSelect()</td>
<td>8</td>
</tr>
<tr>
<td>RvSipStackSelectUntil()</td>
<td>9</td>
</tr>
<tr>
<td>RvSipStackSetAppHandle()</td>
<td>13</td>
</tr>
<tr>
<td>RvSipStackSetNewLogFilters()</td>
<td>25</td>
</tr>
<tr>
<td>RvSipStackStatistics</td>
<td>760</td>
</tr>
<tr>
<td>RvSipSubsAuthBegin()</td>
<td>473</td>
</tr>
<tr>
<td>RvSipSubsAuthCompletedEvt()</td>
<td>916</td>
</tr>
<tr>
<td>RvSipSubsAuthCredentialsFoundEvt()</td>
<td>914</td>
</tr>
<tr>
<td>RvSipSubsAuthProceed()</td>
<td>474</td>
</tr>
<tr>
<td>RvSipSubsAutoRefresh()</td>
<td>429</td>
</tr>
<tr>
<td>RvSipSubsCreatedDueToForkingEvt()</td>
<td>911</td>
</tr>
<tr>
<td>RvSipSubsCreatedEvt()</td>
<td>903</td>
</tr>
<tr>
<td>RvSipSubsCreateNotify()</td>
<td>462</td>
</tr>
<tr>
<td>RvSipSubscriptionType</td>
<td>890</td>
</tr>
<tr>
<td>RvSipSubsDetachOwner()</td>
<td>433</td>
</tr>
<tr>
<td>RvSipSubsDialogInit()</td>
<td>418</td>
</tr>
<tr>
<td>RvSipSubsDialogInitStr()</td>
<td>420</td>
</tr>
<tr>
<td>RvSipSubsDNSContinue()</td>
<td>483</td>
</tr>
<tr>
<td>RvSipSubsDNSGetList()</td>
<td>485</td>
</tr>
<tr>
<td>RvSipSubsDNSGiveUp()</td>
<td>482</td>
</tr>
<tr>
<td>RvSipSubsDNSResendRequest()</td>
<td>484</td>
</tr>
<tr>
<td>RvSipSubsEventPackageType</td>
<td>896</td>
</tr>
<tr>
<td>RvSipSubsEvdHandlers</td>
<td>899</td>
</tr>
<tr>
<td>RvSipSubsExpirationAlertEvt</td>
<td>906</td>
</tr>
<tr>
<td>RvSipSubsFinalDestResolvedEvt</td>
<td>927</td>
</tr>
<tr>
<td>RvSipSubsGetActiveSubsStorageSize()</td>
<td>499</td>
</tr>
<tr>
<td>RvSipSubsGetAppSubsHandle()</td>
<td>444</td>
</tr>
<tr>
<td>RvSipSubsGetCurrentState()</td>
<td>446</td>
</tr>
<tr>
<td>RvSipSubsGetDialogObj()</td>
<td>442</td>
</tr>
<tr>
<td>RvSipSubsGetEventHeader()</td>
<td>438</td>
</tr>
<tr>
<td>RvSipSubsGetEventPackageType()</td>
<td>451</td>
</tr>
<tr>
<td>RvSipSubsGetExpiresVal()</td>
<td>440</td>
</tr>
<tr>
<td>RvSipSubsGetForkingEnabledFlag()</td>
<td>455,</td>
</tr>
<tr>
<td>RvSipSubsGetNewHeaderHandle()</td>
<td>452</td>
</tr>
<tr>
<td>RvSipSubsGetOriginalSubs()</td>
<td>456,</td>
</tr>
<tr>
<td>RvSipSubsGetOutboundMsg()</td>
<td>449</td>
</tr>
<tr>
<td>RvSipSubsGetReceivedMsg()</td>
<td>448</td>
</tr>
<tr>
<td>RvSipSubsGetReferFinalStatus()</td>
<td>497</td>
</tr>
<tr>
<td>RvSipSubsGetReferNotifyStatus()</td>
<td>497</td>
</tr>
<tr>
<td>RvSipSubsGetReferredByHeader()</td>
<td>496</td>
</tr>
<tr>
<td>RvSipSubsGetReferToHeader()</td>
<td>495</td>
</tr>
<tr>
<td>RvSipSubsGetRequestedExpiresVal()</td>
<td>441</td>
</tr>
<tr>
<td>RvSipSubsGetStackInstance()</td>
<td>450</td>
</tr>
<tr>
<td>RvSipSubsGetSubsMgr()</td>
<td>445</td>
</tr>
<tr>
<td>RvSipSubsGetSubsType()</td>
<td>447</td>
</tr>
<tr>
<td>RvSipSubsGetTranscCurrentLocalAddress()</td>
<td>457</td>
</tr>
<tr>
<td>RvSipSubsHandle</td>
<td>878</td>
</tr>
<tr>
<td>RvSipSubsInit()</td>
<td>414</td>
</tr>
<tr>
<td>RvSipSubsInitStr()</td>
<td>416</td>
</tr>
<tr>
<td>RvSipSubsIsOutOfBand()</td>
<td>436</td>
</tr>
<tr>
<td>RvSipSubsMgrCreateOutOfBandSubscription()</td>
<td>408</td>
</tr>
<tr>
<td>RvSipSubsMgrCreateSubscription()</td>
<td>406</td>
</tr>
<tr>
<td>RvSipSubsMgrGetAppMgrHandle()</td>
<td>411</td>
</tr>
<tr>
<td>RvSipSubsMgrGetStackInstance()</td>
<td>410</td>
</tr>
<tr>
<td>RvSipSubsMgrHandle</td>
<td>877</td>
</tr>
<tr>
<td>RvSipSubsMgrSetAppMgrHandle()</td>
<td>412</td>
</tr>
<tr>
<td>RvSipSubsMgrSetEvdHandlers()</td>
<td>405</td>
</tr>
<tr>
<td>RvSipSubsMsgReceivedEvt()</td>
<td>920</td>
</tr>
</tbody>
</table>
RvSipTransmitterMgrGetAppMgrHandle() 270
RvSipTransmitterMgrGetStackInstance() 271
RvSipTransmitterMgrHandle 811
RvSipTransmitterMgrRegExpResolutionNeededEv() 829
RvSipTransmitterMgrSetActiveAppMgrHandle() 269
RvSipTransmitterMgrSetEvHandlers() 272
RvSipTransmitterNewConnInUseStateInfo 818
RvSipTransmitterOtherURLAddressFoundEv() 826
RvSipTransmitterReason 816
RvSipTransmitterRegExpFlag 819
RvSipTransmitterRegExpMatch 820
RvSipTransmitterRegExpResolutionParams 821
RvSipTransmitterResources 1060
RvSipTransmitterResumeSending() 277
RvSipTransmitterSendBuffer() 279
RvSipTransmitterSendMessage() 274
RvSipTransmitterSetAppHandle() 294
RvSipTransmitterSetConnection() 290
RvSipTransmitterSetDestAddress() 302
RvSipTransmitterSetFixViaBranch() 292
RvSipTransmitterSetForceOutOfBandAddrFlag() 307
RvSipTransmitterSetIgnoreOutOfBandProxyFlag() 304
RvSipTransmitterSetKeepMsgFlag() 293
RvSipTransmitterSetLocalAddress() 281
RvSipTransmitterSetOutOfBandDetails() 285
RvSipTransmitterSetPersistency() 288
RvSipTransmitterSetUseFirstRouteFlag() 305
RvSipTransmitterSetViaBranch() 306
RvSipTransmitterState 815
RvSipTransmitterStateChangedEv() 824
RvSipTransmitterTerminate() 278
RvSipTransportAddr 969
RvSipTransportAddressType 945
RvSipTransportBadSyntaxStartLineMsgEv() 988
RvSipTransportBadSyntaxStartLineMsgEv() 990
RvSipTransportBsAction 960
RvSipTransportBufferReceivedEv() 994
RvSipTransportBufferToSentEv() 996
RvSipTransportConnectionAppHandle 939
RvSipTransportConnectionAttachOwner() 534
RvSipTransportConnectionCfg 948
RvSipTransportConnectionCreate() 532
RvSipTransportConnectionCreatedEv() 992
RvSipTransportConnectionDataReceivedEv() 1005
RvSipTransportConnectionDetachOwner() 535
RvSipTransportConnectionDisable() 537
RvSipTransportConnectionEnable() 536
RvSipTransportConnectionEnableConnByAlias() 566
RvSipTransportConnectionEvHandlers 965, 973
RvSipTransportConnectionGetAlias() 567
RvSipTransportConnectionGetAppHandle() 549
RvSipTransportConnectionGetCurrentState() 539
RvSipTransportConnectionGetCurrentTlsState() 540
RvSipTransportConnectionGetIpTosSockOption() 553, 561
RvSipTransportConnectionGetLocalAddress() 543
RvSipTransportConnectionGetNumOfOwners() 541
RvSipTransportConnectionGetRemoteAddress() 544
RvSipTransportConnectionGetStackInstance() 545
RvSipTransportConnectionGetTransportType() 542
RvSipTransportConnectionHandle 937
RvSipTransportConnectionInit() 531
RvSipTransportConnectionIsEnabled() 538
RvSipTransportConnectionOwnerHandle 938
RvSipTransportConnectionParserResultEv() 998
RvSipTransportConnectionServerReuseEv() 1007
RvSipTransportConnectionSetAppHandle() 550
RvSipTransportConnectionSetIpTosSockOption() 552, 560
RvSipTransportConnectionState 950
RvSipTransportConnectionStateChangedEv() 979
RvSipTransportConnectionStateChangedReason 955
RvSipTransportConnectionStatus 953
RvSipTransportConnectionStatusEv() 981
RvSipTransportConnectionTerminate() 533
RvSipTransportConnectionTlsGetEncodedCertificate() 551
RvSipTransportConnectionTlsGetUnderlyingSsl() 568
RvSipTransportConnectionTlsHandshake() 546
RvSipTransportConnectionTlsPostConnectionAssertionEv() 985
RvSipTransportConnectionTlsRenegotiate() 548, 569
RvSipTransportConnectionTlsSequenceStartedEv() 983
RvSipTransportConnectionTlsState 951
RvSipTransportConnectionTlsStateChangedEv() 984
RvSipTransportConnectionTlsStatus 954
RvSipTransportConnectionTlsStatusEv() 1002
RvSipTransportConnectionType 972
RvSipTransportConvertIpToString() 528
RvSipTransportConvertStringToIp() 527
RvSipTransportDnsGetEnumResult() 586
RvSipTransportDNSHostNameElement 976
RvSipTransportDNSIPElement 977
RvSipTransportDNSListConstruct() 597
RvSipTransportDNSListDestruct() 598
RvSipTransportDNSListGetHostElement() 585
RvSipTransportDNSListGetIPElement() 590
RvSipTransportDNSListGetSrvElement() 580, 599
RvSipTransportDNSListGetUsedHostElement() 593
RvSipTransportDNSListGetUsedSrvElement() 594
RvSipTransportDNSListPopHostElement() 587
RvSipTransportDNSListPopIPElement() 591
RvSipTransportDNSListPopSrvElement() 582
RvSipTransportDNSListPushHostElement() 583
RvSipTransportDNSListPushIPElement() 588
RvSipTransportDNSListPushSrvElement() 578
RvSipTransportDNSListRemoveTopmostHostElement() 584
RvSipTransportDNSListRemoveTopmostIPElement() 589
RvSipTransportDNSListRemoveTopmostSrvElement() 579
RvSipTransportDNSListSetUsedHostElement() 596
RvSipTransportDNSListSetUsedSrvElement() 595
RvSipTransportDNSSRVElement 975
RvSipTransportGetIPv4LocalAddressByIndex() 562
RvSipTransportGetIPv6LocalAddress() 563
RvSipTransportGetNumberOfDNSListEntries() 592
RvSipTransportGetNumberOfIPv4LocalAddresses() 564
RvSipTransportInjectMsg() 529
RvSipTransportLocalAddrAppHandle 942
RvSipTransportLocalAddrHandle 941
RvSipTransportMgrCreateConnection() 516
RvSipTransportMgrEvHandlers 961
RvSipTransportMgrGetAppMgrHandle() 513
RvSipTransportMgrGetStackInstance() 515
RvSipTransportMgrHandle 935
RvSipTransportMgrLocalAddressAdd() 518