SIP RECORDING SDK

TECHNICAL DOCUMENTATION

VERSION 8.0.4.2

CONTENTS

INTRODUCTION AND QUICK START

The VaxVoIP SIP CALL RECORDING COM (Component Object Model) component, specifically the VaxTeleServerREC.dll, serves as a robust framework comprising various functions and events tailored for the seamless development of SIP REC (Session Initiation Protocol Recording) Protocol-based Call Recording Servers.

This dynamic VaxVoIP SIP REC COM component, encapsulated within the VaxTeleServerREC.dll, empowers developers by providing a comprehensive set of functions and events. This facilitates the swift and efficient creation of SIP REC-based Call Recording Servers, offering versatility for integration in diverse environments such as call centers, offices, and other SIP-based VoIP services.

By leveraging the capabilities of the VaxVoIP SIP REC COM component, developers gain a powerful toolset to enhance call recording functionalities, ensuring compatibility with SIP protocols and optimizing performance across various VoIP services.

EXPORTED FUNCTIONS

SetLicenseKey()

The trial version of the VaxVoIP SDK offers a 30-day evaluation period. After this period, a license key is required to continue using the SDK without interruption and to remove the evaluation message box that appears during usage. License keys are provided upon purchase.

To convert the trial version into a fully registered version with no expiration or trial limitations, use the SetLicenseKey() method. By invoking this method with a valid license key, the SDK will function without any time restrictions.

Syntax

```
SetLicenseKey(LicenseKey)
```

Parameters

LicenseKey (string)
The value of this parameter is the license key provided by VaxVoIP.

Return Value

No return value.

Example

```
SetLicenseKey("LicenseKey")
Initialize("")
```

See Also

Initialize(), GetVaxErrorCode()

GetVaxErrorCode()

The GetVaxErrorCode() method retrieves the error code associated with the last failed operation. This error code provides detailed information about the nature of the failure, aiding in debugging and troubleshooting.

For a detailed explanation of the possible error codes and their meanings, please refer to the **LIST OF ERROR CODES** section.

Syntax

```
integer GetVaxErrorCode()
```

Parameters

No parameters.

Return Value

The GetVaxErrorCode() returns the error code.

Example

```
SetLicenseKey("LicenseKey")

Result = Initialize("")
if(Result == 0) GetVaxErrorCode()
```

See Also

OnVaxErrorLog(), OnCallSessionErrorLog()

Initialize()

he Initialize() function sets up the VaxVoIP SIP REC Server COM component. This process includes allocating internal memory resources and configuring the component for integration with your application.

Once executed, the component becomes fully operational, exposing its methods for external use. This setup ensures that your application can effectively interact with the component and utilize its various functionalities.

Syntax

```
boolean Initialize(DomainRealm)
```

Parameters

DomainRealm (string)

This parameter value is used internally to generate SIP URIs and is included as the "realm" field in SIP packets during the authentication of SIP clients (such as softphones and hardphones) and call processing.

Parameter Purpose: The parameter dictates the domain used for SIP URIs and authentication. It is utilized as the "realm" field in SIP packets.

Blank/Empty Value: If this parameter is left blank or set to an empty string, the SIP authentication process and SIP URI generation will use the assigned IP address instead.

Examples: With DomainRealm Set: If the DomainRealm value is set to sip.vaxvoip.com, VaxVoIP will create SIP URIs in the format sip:username@sip.vaxvoip.com.
With Empty DomainRealm: If the DomainRealm is set to an empty string and the assigned IP address is 10.3.5.66, VaxVoIP will generate SIP URIs in the format sip:username@10.3.5.66.

Note: It is not mandatory for an IP address to be assigned to the DomainRealm parameter.

Return Value

Upon successful execution, this function returns a non-zero value; otherwise, it returns 0. To obtain specific error details, the GetVaxErrorCode() method can be invoked.

Example

```
Initialize("")

or

Initialize("demo.vaxvoip.com")

or

Initialize("vaxvoip.com")
```

See Also

UnInitialize(), GetVaxErrorCode(), SetListenPortRangeRTP()

UnInitialize()

The UnInitialize() function releases all resources allocated by the Initialize() function. Its primary purpose is to ensure a clean and orderly release of resources when they are no longer needed, preventing resource leaks and maintaining system stability.

Syntax

```
UnInitialize()
```

Parameters

No parameters.

Return Value

No return value.

Example

```
UnInitialize()
```

See Also

Initialize()

OpenNetworkUDP()

The OpenNetworkUDP() function initializes a UDP socket, assigns a listening port, and begins monitoring for incoming SIP requests. It also manages the sending of SIP responses over UDP. Based on the received SIP requests, the relevant COM (Component Object Model) events are triggered accordingly.

Syntax

```
boolean OpenNetworkUDP(
                        ListenIP,
                        ListenPort
                        )
```

Parameters

ListenIP (string)
: This parameter specifies the IP address on which the VaxVoIP SIP REC Server listens for incoming SIP requests over UDP. It can be set to an empty value or an IP address assigned to the computer where the VaxVoIP COM-integrated SIP server is running.

ListenPort (integer)
: This parameter defines the port number on which the SIP server receives SIP requests. The standard SIP listening port is 5060.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
OpenNetworkUDP("", 5060)

or

OpenNetworkUDP("192.168.0.3", 5060)
```

See Also

OpenNetworkTCP(), OpenNetworkTLS(), CloseNetworkUDP(), CloseNetworkTCP(), CloseNetworkTLS(), GetVaxErrorCode(), SetListenPortRangeRTP()

OpenNetworkTCP()

The OpenNetworkTCP() function sets up a TCP socket, assigns a listening port, and starts monitoring for incoming SIP requests. It also manages the sending of SIP responses over TCP. Upon receiving SIP requests, the corresponding COM (Component Object Model) events are triggered.

Syntax

```
boolean OpenNetworkTCP(
                        ListenIP,
                        ListenPort
                        )
```

Parameters

ListenIP (string)
: This parameter specifies the IP address on which the VaxVoIP SIP REC Server listens for incoming SIP requests over TCP. It can be set to either an empty value or an IP address assigned to the computer running the VaxVoIP COM-integrated SIP server.

ListenPort (integer)
: This parameter defines the port number on which the SIP server accepts SIP requests. The default SIP listening port is 5060.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
OpenNetworkTCP("", 5060)

or

OpenNetworkTCP("192.168.0.3", 5060)
```

See Also

OpenNetworkUDP(), OpenNetworkTLS(), CloseNetworkUDP(), CloseNetworkTCP(), CloseNetworkTLS(), GetVaxErrorCode(), SetListenPortRangeRTP()

OpenNetworkTLS()

The OpenNetworkTLS() function establishes a secure TLS communication channel, allocates a TLS listening port, and starts monitoring for incoming SIP requests over TLS. It also manages the sending of outgoing SIP responses. Upon receiving SIP requests, the relevant COM (Component Object Model) events are triggered.

Syntax

```
boolean OpenNetworkTLS(
                        ListenIP,
                        ListenPort,
                        CertPEM
                        )
```

Parameters

ListenIP (string)
: This parameter specifies the IP address on which the VaxVoIP SIP REC Server listens for incoming SIP requests over TLS. It can be an empty value or an IP address assigned to the computer running the VaxVoIP COM-integrated SIP server.

ListenPort (integer)
: This parameter defines the port number on which the SIP server receives SIP requests over TLS. The standard SIP listening port for TLS is 5061.

CertPEM (string)
: This parameter specifies the filename or data of the SSL certificate in PEM format. It can be provided as a string or text value. If it is empty, VaxVoIP COM uses the default VaxVoIP SSL certificate.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
OpenNetworkTLS("", 5061, "")

or

OpenNetworkTLS("192.168.0.3", 5061, "")
```

See Also

OpenNetworkUDP(), OpenNetworkTCP(), CloseNetworkUDP(), CloseNetworkTCP(), CloseNetworkTLS(), GetVaxErrorCode(), SetListenPortRangeRTP()

CloseNetworkUDP()

The CloseNetworkUDP() function closes a UDP socket, terminating its operation and releasing all associated network resources. This function ensures that no further data transmission or reception occurs on the socket, effectively disconnecting it from the network.

Syntax

```
CloseNetworkUDP()
```

Parameters

No parameters.

Return Value

No return value.

Example

```
OpenNetworkUDP("", 5060)
CloseNetworkUDP()
```

See Also

OpenNetworkUDP(), OpenNetworkTCP(), OpenNetworkTLS(), CloseNetworkTCP(), CloseNetworkTLS(), GetVaxErrorCode(), SetListenPortRangeRTP()

CloseNetworkTCP()

The CloseNetworkTCP() function closes a TCP socket, terminating the connection and releasing all associated resources. This function ensures a proper shutdown of the TCP connection, preventing further data transmission or reception. By closing the socket, it helps free up system resources and maintain network stability.

Syntax

```
CloseNetworkTCP()
```

Parameters

No parameters.

Return Value

No return value.

Example

```
OpenNetworkUDP("", 5060)
OpenNetworkTCP("", 5060)

CloseNetworkTCP()
CloseNetworkUDP()
```

See Also

OpenNetworkUDP(), OpenNetworkTCP(), OpenNetworkTLS(), CloseNetworkUDP(), CloseNetworkTLS(), GetVaxErrorCode(), SetListenPortRangeRTP()

CloseNetworkTLS()

The CloseNetworkTLS() function securely terminates a TLS communication channel, closing the underlying socket and releasing all associated resources. This function ensures that the secure connection is properly shut down, preventing any further encrypted data transmission or reception. By closing the TLS channel, it helps maintain network security and integrity while freeing up system resources.

Syntax

```
CloseNetworkTLS()
```

Parameters

No parameters.

Return Value

No return value.

Example

```
OpenNetworkUDP("", 5060)
OpenNetworkTCP("", 5060)
OpenNetworkTLS("", 5061, "")

CloseNetworkTLS()
CloseNetworkTCP()
CloseNetworkUDP()
```

See Also

OpenNetworkUDP(), OpenNetworkTCP(), OpenNetworkTLS(), CloseNetworkUDP(), CloseNetworkTCP(), GetVaxErrorCode(), SetListenPortRangeRTP()

SetListenPortRangeRTP()

The SetListenPortRangeRTP() function sets the specified port range for VaxVoIP to allocate and open the RTP listening port.

The port must comply with SIP RFC 2327, which requires it to be within the range of 1024 to 65535 and an even number to ensure RTP compliance.

Syntax

```
boolean SetListenPortRangeRTP(ListenStartPort, ListenEndPort)
```

Parameters

ListenStartPort (integer)
: This parameter defines the starting port of the specified range for the RTP listen port.

ListenEndPort (integer)
: This parameter defines the ending port of the specified range for the RTP listen port.

Return Value

The function returns a non-zero value upon successful execution; otherwise, it returns 0. In case of failure, a specific error code can be retrieved by calling the GetVaxErrorCode() method.

Example

```
Result = Initialize("")
if(Result == 0) GetVaxErrorCode()

OpenNetworkUDP("", 5060)

SetListenPortRangeRTP(20000, 40000)
```

See Also

Initialize(), GetVaxErrorCode()

AddNetworkRouteSIP(), AddNetworkRouteRTP()

The AddNetworkRouteSIP() and AddNetworkRouteRTP() functions allow the addition of supplementary IP addresses to facilitate the routing of SIP and RTP packets, respectively. These functions are particularly useful in network configurations where the VaxVoIP integrated Server REC.SIP is positioned behind a firewall, router, or NAT with port forwarding enabled.

In such scenarios, it is recommended to initialize VaxVoIP with the private IP address assigned to the computer. By doing so, the server can communicate effectively within the local network.

To ensure proper routing of SIP and RTP packets through the public network, the AddNetworkRouteSIP() and AddNetworkRouteRTP() functions should be used to incorporate the public IP address assigned to the router.

This setup ensures that the VaxVoIP REC.server can handle incoming and outgoing traffic correctly, maintaining seamless communication despite the presence of NAT or other network barriers.

Syntax

```
boolean AddNetworkRouteSIP(AssignedIP, RouterIP)
boolean AddNetworkRouteRTP(AssignedIP, RouterIP)
```

Parameters

AssignedIP (string)
: This parameter specifies the private IP address assigned to the computer within the local network.

RouterIP (string)
: This parameter specifies the public IP address assigned to the router, used for external network communication.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
Result = Initialize("")
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkUDP("192.168.0.25", 5060)
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkTCP("192.168.0.25", 5060)
if(Result = 0) GetVaxErrorCode()

AddNetworkRouteSIP("192.168.0.25", "66.77.88.99")
AddNetworkRouteRTP("192.168.0.25", "66.77.88.99")

SetListenPortRangeRTP(10000, 20000)
```

To enable port forwarding and ensure the SIP server functions correctly behind a firewall or router, follow these steps in your router's settings:

Forward Inbound UDP Ports 10000 to 20000: Configure the router to forward incoming UDP traffic on ports 10000 to 20000 to the computer running the SIP server.

Enable Outbound UDP Ports 1024 to 65535: Allow outbound UDP traffic on ports 1024 to 65535 to ensure that the SIP server can communicate with external networks.

Assign IP Addresses: Ensure that the IP address 192.168.0.25 is assigned to the computer behind the router and 66.77.88.99 is the public IP address assigned to the router.

This setup will enable the SIP server to operate effectively behind the firewall, allowing proper routing of SIP and RTP packets.

See Also

Initialize(), SetListenPortRangeRTP(), GetVaxErrorCode(), OpenNetworkUDP(), OpenNetworkTCP(), OpenNetworkTLS()

AddUser()

The AddUser() function adds users to the SIP REC server developed using the VaxVoIP COM component (VaxTeleServerREC.dll). This function is essential for managing SIP user accounts, integrating users into the server's internal system.

When invoked, AddUser() creates and maintains a list of users that the SIP REC server uses for handling SIP registration and call requests. This user list is crucial for processing incoming connection requests, authenticating users, and managing user sessions.

By using the AddUser() function, you enable users to log in and register with the VaxVoIP-based SIP REC server. Usernames and passwords can then be used by SIP-based softphones or hardphones to connect to and interact with the server, facilitating effective communication and integration within the SIP network.

Syntax

```
boolean AddUser(
                UserName,
                Password,
                AudioCodecList
                )
```

Parameters

UserName (string)
: This parameter specifies the user's login name.

Password (string)
: This parameter specifies the password associated with the user's login.

AudioCodecList (string)
: This parameter specifies the list of audio codecs to be used in the call request.

  0 = GSM
  1 = iLBC
  2 = G711 A-Law
  3 = G711 U-Law
  4 = G729

  "03" indicates that only GSM and G711 U-Law codecs are supported for the call, with GSM having a higher priority.

  "4213" indicates that the supported codecs for the call request are G729, G711 A-Law, iLBC, and G711 U-Law. In this case, G729 has the highest priority (4), and G711 U-Law has the lowest priority (3).

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
SetLicenseKey("LicenseKey")

Result = Initialize("")
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkUDP("", 5060)
if(Result = 0) GetVaxErrorCode()

// 3 = G711 U-Law, 2 = G711 A-Law, 4 = G729
// (G711 U-Law has the highest priority, G729 has the lowest priority)

Result = AddUser("9090", "123", "324")
if(Result == 0) GetVaxErrorCode()

OnRegisterUser(UserName, Domain, UserAgentName, FromIP, FromPort,
                    RegId)
{
        AuthRegister(RegId)
}
```

```
SetLicenseKey("LicenseKey")

Result = Initialize("")
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkUDP("", 5060)
if(Result = 0) GetVaxErrorCode()

OnRegisterUser(UserName, Domain, UserAgentName, FromIP, FromPort,
                RegId)
{
        // Fetch UserName details from storage (RecordDB)

        Result = AddUser(UserName, RecordDB.Password,
                        RecordDB.AudioCodec)

        if(Result == 0) GetVaxErrorCode()

        AuthRegister(RegId)
}

OnRegisterUserFailed(UserName, FromIP, FromPort, RegId)
{
        RemoveUser(UserName)
}

OnUnRegisterUser(UserName)
{
        RemoveUser(UserName)
}
```

See Also

RemoveUser(), RejectRegister(), AcceptRegister(), AuthRegister(), GetVaxErrorCode(), OnRegisterUser(), OnUnRegisterUser(), OnRegisterUserFailed(), OnRegisterUserSuccess()

RemoveUser()

The RemoveUser() function deletes a user that was previously added using the AddUser() function.

This function removes a user account from the SIP REC server, ensuring that the user can no longer access or interact with the server.

By calling RemoveUser(), you effectively clean up the user list and free up resources associated with that user account.

Syntax

```
RemoveUser(UserName)
```

Parameters

UserName (string)
: This parameter specifies the user's login name.

Return Value

No return value.

Example

```
Result = Initialize("")
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkUDP("", 5060)
if(Result = 0) GetVaxErrorCode()

OnRegisterUser(UserName, Domain, UserAgentName, FromIP, FromPort,
                RegId)
{
        // Retrieve UserName account details from storage (RecordDB)

        // 2 = G711a, 4 = G729, 3 = G711u
        // (G711a has the highest priority, G711u has the lowest priority)

        Result = AddUser(UserName, RecordDB.AccountPassword, "243")
        if(Result == 0) GetVaxErrorCode()

        AuthRegister(RegId)
}

 OnRegisterUserFailed(UserName, FromIP, FromPort, RegId)
{
        RemoveUser(UserName)
}

OnUnRegisterUser(UserName)
{
        RemoveUser(UserName)
}
```

See Also

AddUser(), RejectRegister(), AcceptRegister(), AuthRegister(), GetVaxErrorCode(), OnRegisterUser(), OnUnRegisterUser(), OnRegisterUserFailed(), OnRegisterUserSuccess()

RegisterUserExpiry()

The RegisterUserExpiry() function configures the expiration interval for SIP REC.client registrations in the VaxVoIP SIP REC Server. This setting determines how long the registration information for a SIP client remains valid before requiring renewal.

If a value of -1 is set, the SIP REC Server accepts the expiration interval requested by the SIP client during the registration process.

If a value other than -1 is set, the SIP REC Server ignores the client's requested expiration interval and uses the value specified by RegisterUserExpiry() instead. This allows for customized control over the expiration interval.

If RegisterUserExpiry() is not explicitly called, the SIP REC Server defaults to an expiration interval of 30 seconds, which it communicates to the SIP client(s) during registration.

The SIP packet's Expires header designates the expiration interval of the registration.

Syntax

```
boolean RegisterUserExpiry(Expiry)
```

Parameters

Expiry (integer)
: This parameter specifies the time interval, in seconds, for the expiration of the registration.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
Result = Initialize("")
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkUDP("", 5060)
if(Result = 0) GetVaxErrorCode()

RegisterUserExpiry(1800)
```

See Also

AddUser(), OnRegisterUser(), AcceptRegister(), AuthRegister()

AcceptRegister()

The AcceptRegister() function processes and accepts registration requests received from SIP REC.clients, such as softphones and hardphones.

During the SIP registration process, SIP clients send registration requests to the SIP REC server. The server is responsible for authorizing the provided login credentials and then accepting these requests.

When VaxVoIP receives a registration request, it triggers the OnRegisterUser() event. Within this event, calling the AcceptRegister() function accepts the registration request without performing any authorization of the login credentials.

In essence, the AcceptRegister() function bypasses the login authorization process, directly accepting the registration request as-is. This function is used when you want to allow registration requests to be accepted without validating user credentials.

For additional information, refer to the SIP CLIENT REGISTRATION PROCESS

Syntax

```
boolean AcceptRegister(RegId)
```

Parameters

RegId (integer)
This parameter identifies a unique registration session.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
SetLicenseKey("LicenseKey")

Result = Initialize("")
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkUDP("", 5060)
if(Result = 0) GetVaxErrorCode()

OnRegisterUser(UserName, Domain, UserAgentName, FromIP, FromPort,
                RegId)
{
        // Fetch UserName details from storage (RecordDB)

        Result = AddUser(UserName, RecordDB.Password,
                        RecordDB.AudioCodec)

        if(Result == 0) GetVaxErrorCode()

        AcceptRegister(RegId)
}

OnRegisterUserFailed(UserName, FromIP, FromPort, RegId)
{
        RemoveUser(UserName)
}

OnUnRegisterUser(UserName)
{
        RemoveUser(UserName)
}
```

See Also

AddUser(), RemoveUser(), RejectRegister(), AuthRegister(), GetVaxErrorCode(), OnRegisterUser(), OnUnRegisterUser(), OnRegisterUserFailed(), OnRegisterUserSuccess()

RejectRegister()

The RejectRegister() function processes and rejects registration requests from SIP REC clients directed to VaxVoIP.

When VaxVoIP receives a registration request, it triggers the OnRegisterUser() event. If the RejectRegister() function is invoked during this event, it denies the registration request, preventing the client from registering with the SIP REC server.

This function ensures that the SIP REC server can effectively handle invalid or unauthorized registration attempts.

For further details, refer to the **LIST OF SIP RESPONSES**

Syntax

```
boolean RejectRegister(
                        RegId,
                        StatusCode,
                        ReasonPhrase
                        )
```

Parameters

RegId (integer)
: This parameter specifies a unique identifier for a registration session.

StatusCode (integer)
: This parameter specifies the SIP response status code.

ReasonPhrase (string)
: This parameter specifies the SIP response reason phrase, such as Unauthorized or Not Found.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
SetLicenseKey("LicenseKey")

Result = Initialize("")
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkUDP("", 5060)
if(Result = 0) GetVaxErrorCode()

OnRegisterUser(UserName, Domain, UserAgentName, FromIP, FromPort,
                RegId)
{
        // Fetch UserName details from storage (RecordDB)

        if(RecordDB not found)
        {
                RejectRegister(UserName, 404, "Not Found")
        }
        else
        {
                AddUser(UserName, RecordDB.Password, "32")
                AuthRegister(RegId)
        }
}

OnRegisterUserFailed(UserName, FromIP, FromPort, RegId)
{
        RemoveUser(UserName)
}

OnUnRegisterUser(UserName)
{
        RemoveUser(UserName)
}
```

See Also

AddUser(), RemoveUser(), AuthRegister(), AcceptRegister(), GetVaxErrorCode(), OnRegisterUser(), OnUnRegisterUser(), OnRegisterUserFailed(), OnRegisterUserSuccess()

AuthRegister()

The AuthRegister() function initiates the authentication process for a specified user before accepting the registration request.

SIP REC clients send SIP REGISTER requests to connect and register with SIP REC servers.

When VaxVoIP receives a registration request, it triggers the OnRegisterUser() event and starts the authentication process by calling the AuthRegister() function.

Upon calling the AuthRegister() function, VaxVoIP:

1. Starts the SIP authentication process.
2. Completes the authentication process.
3. Accepts the registration request.
4. Triggers the OnRegisterUserSuccess() event if authentication is successful or the OnRegisterUserFailed() event if authentication fails.

Please see SIP CLIENT REGISTRATION PROCESS for more details.

Syntax

```
boolean AuthRegister(RegId)
```

Parameters

RegId (integer)
: This parameter specifies a unique identification of a registration session.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
SetLicenseKey("LicenseKey")

Result = Initialize("")
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkUDP("", 5060)
if(Result = 0) GetVaxErrorCode()

OnRegisterUser(UserName, Domain, UserAgentName, FromIP, FromPort,
                RegId)
{
        // Fetch UserName details from storage (RecordDB)

        if(RecordDB not found)
        {
                RejectRegister(UserName, 404, "Not Found")
        }
        else
        {
                AddUser(UserName, RecordDB.Password, "32")
                AuthRegister(RegId)
        }
}

OnRegisterUserSuccess(UserName, FromIP, FromPort, RegId)
{

}

OnRegisterUserFailed(UserName, FromIP, FromPort, RegId)
{
        RemoveUser(UserName)
}

OnUnRegisterUser(UserName)
{
        RemoveUser(UserName)
}
```

See Also

AddUser(), RemoveUser(), RejectRegister(), AcceptRegister(), GetVaxErrorCode(), OnRegisterUser(), OnUnRegisterUser(), OnRegisterUserFailed(), OnRegisterUserSuccess()

AddLine()

The AddLine() function adds third-party SIP server account settings as a line in VaxVoIP, allowing VaxREC to work with other SIP servers as well.

VaxREC is an SDK for SIP-REC (Session Initiation Protocol Recording) servers. Software created with the VaxREC SDK can connect to other SIP servers and devices using the SIP protocol, facilitating the exchange of SIP-REC call requests with these servers and devices.

To integrate VaxREC with another SIP server, create a user account on the target SIP server. Then, configure VaxREC by adding the user account settings using the AddLine() function. This establishes a connection between VaxREC and the SIP server, enabling seamless communication.

Syntax

```
boolean AddLine(
                LineName,
                LineType
                DisplayName,
                UserName,
                AuthUser
                AuthPwd,
                DomainRealm,
                ServerAddr,
                ServerPort,
                AudioCodecList
                )
```

Parameters

LineName (string)
: This parameter value specifies the unique line name to identify a specific line.

LineType (integer)
: This parameter value specifies the line type.

  0 = Line Type UDP
  1 = Line Type TCP
  2 = Line Type TLS

DisplayName (string)
: This Parameter value specifies the display name for user which is provided by IP-Telephony service provider or third party SIP server.

UserName (string)
: This Parameter value specifies the user name which is provided by IP-Telephony service provider or third party SIP server.

AuthUser (string)
This Parameter value specifies the user Login which is provided by IP-Telephony service provider or third party SIP server.

AuthPwd (string)
This Parameter value specifies the password which is provided by IP-Telephony service provider or third party SIP server.

DomainRealm (string)
This Parameter value is provided by IP-Telephony service provider or third party SIP server.

ServerAddr (string)
This Parameter value is provided by IP-Telephony service provider or third party SIP server.

ServerPort (integer)
This Parameter value is provided by IP-Telephony service provider or third party SIP server.

AudioCodecList (string)
This parameter specifies the list of audio codecs to be used in the call request.

0 = GSM
1 = iLBC
2 = G711 A-Law
3 = G711 U-Law
4 = G729

"03" indicates that only GSM and G711 U-Law codecs are supported for the call, with GSM having a higher priority.

"4213" indicates that the supported codecs for the call request are G729, G711 A-Law, iLBC, and G711 U-Law. In this case, G729 has the highest priority (4), and G711 U-Law has the lowest priority (3).

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
Result = Initialize("")
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkUDP("", 5060)
if(Result = 0) GetVaxErrorCode()

AddLine("LineDial", 0, "", "", "8034", "9341", "", "192.168.0.3", 5060, "32")
RegisterLine("LineDial", 1800)

OnLineRegisterTrying(LineName)
{
}

OnLineRegisterFailed(LineName, StatusCode, ReasonPhrase)
{
}

OnLineRegisterSuccess(LineName)
{
}
```

See Also

RemoveLine(), RegisterLine(), UnRegisterLine(), GetVaxErrorCode(), OnLineRegisterFailed(), OnLineRegisterSuccess(), OnLineRegisterTrying(), OnLineUnRegisterSuccess(), OnLineUnRegisterFailed()

RemoveLine()

The RemoveLine() function deletes a specified line that was previously added using the AddLine() function.

When you call RemoveLine(), it removes the line configuration from VaxVoIP, which was previously integrated into the system through AddLine(). This action effectively disconnects the SIP account or connection associated with that line, ceasing any interactions or communications that were facilitated by it.

This function is useful for managing and updating your SIP line configurations by allowing the removal of obsolete or unwanted lines.

Syntax

```
RemoveLine(LineName)
```

Parameters

LineName (string)
: This parameter value specifies the unique name assigned to identify a particular line.

Return Value

No return value.

Example

```
Result = Initialize("")
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkUDP("", 5060)
if(Result = 0) GetVaxErrorCode()

AddLine("LineDial", 0, "", "", "8034", "9341", "", "192.168.0.3", 5060, "32")
RegisterLine("LineDial", 1800)

RemoveLine("LineDial")
```

See Also

AddLine(), RegisterLine(), UnRegisterLine(), GetVaxErrorCode(), OnLineRegisterFailed(), OnLineRegisterSuccess(), OnLineRegisterTrying(), OnLineUnRegisterSuccess(), OnLineUnRegisterFailed()

RegisterLine()

The RegisterLine() function is used to register a line with an external third-party SIP-REC supported SIP server. During the line registration process, VaxREC triggers various registration-related events, such as OnLineRegisterTrying(), OnLineRegisterSuccess(), and others.

These events provide feedback and notifications at different stages of the line registration process, enabling effective handling and monitoring.

Before invoking RegisterLine(), it is necessary to add the line using the AddLine() function.

Syntax

```
boolean RegisterLine(
                        LineName,
                        Expire
                        )
```

Parameters

LineName (string)
: This parameter specifies the unique name assigned to identify a specific line within the system.

Expire (integer)
: The Expire parameter specifies the time interval (in seconds) after which the registration with the server will be refreshed. This ensures that the server remains updated about the current status of the client.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
Result = Initialize("")
if(Result = 0) GetVaxErrorCode()

Result = OpenNetworkUDP("", 5060)
if(Result = 0) GetVaxErrorCode()

AddLine("LineDial", 0, "", "", "8034", "9341", "", "192.168.0.3", 5060, "32")
RegisterLine("LineDial", 1800)

OnLineRegisterTrying(LineName)
{
}

OnLineRegisterFailed(LineName, StatusCode, ReasonPhrase)
{
}

OnLineRegisterSuccess(LineName)
{
}
```

See Also

AddLine(), RemoveLine(), UnRegisterLine(), GetVaxErrorCode(), OnLineRegisterFailed(), OnLineRegisterSuccess(), OnLineRegisterTrying(), OnLineUnRegisterSuccess(), OnLineUnRegisterFailed()

UnRegisterLine()

The UnRegisterLine() function unregisters the line that was previously registered using the RegisterLine() function. During the unregistration process, VaxREC triggers various related events, such as OnLineUnRegisterTrying() and OnLineUnRegisterSuccess().

Syntax

```
boolean UnRegisterLine(LineName)
```

Parameters

LineName (string)
: Specifies the unique name assigned to identify a particular line within the system.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
UnRegisterLine("LineDial")
```

See Also

RegisterLine(), AddLine(), OnLineUnRegisterSuccess(), GetVaxErrorCode() OnLineUnRegisterTrying(), OnLineUnRegisterFailed()

AcceptCallSession()

The AcceptCallSession() function is used to accept an incoming SIP-REC call request.

The OnIncomingCall() event triggers when the VaxREC-based SIP REC Server receives an incoming call request. This event calls the AcceptCallSession() method to process the incoming call request, facilitating the acceptance and handling of the call session.

Syntax

```
boolean AcceptCallSession(
                              SessionId,
                              Timeout
                              )
```

Parameters

SessionId (integer)
: This parameter specifies a unique identifier for a call session.

Timeout (integer)
: This parameter specifies the time interval, in seconds, during which VaxREC waits for the call session to be connected or established. If the call session is not connected within the specified time interval, a timeout occurs, triggering the OnCallSessionTimeout() event. This event notifies you when a timeout occurs during the establishment of a call session.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
OnIncomingCall(SessionId, CallerName, CallerId, CalleeName, CalleeId,
                    DialNo, FromPeerType, FromPeerName, UserAgentName,
                    RecXML, FromIP, FromPort)
{
        AcceptCallSession(SessionId, 20)
}
```

See Also

RejectCallSession(), CloseCallSession(), GetVaxErrorCode(), OnIncomingCall(), OnCallSessionCreated(), OnCallSessionClosed()

RejectCallSession()

The RejectCallSession() function rejects an incoming call request associated with a specific Call-Session.

Syntax

```
boolean RejectCallSession(
                        SessionId,
                        StatusCode,
                        ReasonPhrase
                        )
```

Parameters

SessionId (integer)
: This parameter specifies a unique identifier for a call session.

StatusCode (integer)
: This parameter specifies the SIP response status.
For further details, refer to the **LIST OF SIP RESPONSES**

ReasonPhrase (string)
: Specifies the SIP response reason phrase, such as "Unauthorized" or "Not Found."

Return Value

Upon successful execution, the function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by calling the GetVaxErrorCode() method.

Example

```
OnIncomingCall(SessionId, CallerName, CallerId, CalleeName, CalleeId,
            DialNo, FromPeerType, FromPeerName, UserAgentName,
            RecXML, FromIP, FromPort)

{
    RejectCallSession(SessionId)
}
```

See Also

AcceptCallSession(), CloseCallSession(), OnIncomingCall(), OnCallSessionClosed()

CloseCallSession()

The CloseCallSession() function terminates the connection, disconnecting all calls within that session. This function is particularly useful in scenarios where the VaxVoIP SDK-integrated SIP REC Server needs to disconnect a call session

Syntax

```
boolean CloseCallSession(SessionId)
```

Parameters

SessionId (integer)
Specifies a unique identifier for a call session.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
CloseCallSession(SessionId)
```

See Also

RejectCallSession(), AcceptCallSession(), OnCallSessionClosed()

AccessAudioPCM()

AccessAudioPCM() provides real-time access to raw audio PCM data, essential for applications integrated with VaxVoIP where real-time audio processing is required. This includes AI and machine learning tasks, speech recognition, language translation, text-to-speech, and speech-to-text operations. The SDK supports uncompressed 8kHz, 16-bit, Mono PCM format.

Named-Pipe Method: This method uses a named pipe to facilitate communication between a pipe server and a pipe client, which can occur either on the same computer or across different computers over a network. In this approach, VaxVoIP internally creates a pipe server using the specified `PipeName` parameter, while the application creates a pipe client to connect to this server. Once connected, the pipe server delivers PCM audio data to the pipe client. This method is recommended for large-scale audio processing in multi-threaded environments, such as recording (REC) servers managing a high volume of VoIP calls. For more details on named pipes, refer to the Microsoft MSDN documentation.

Event-Driven Method: The event-driven approach offers a simpler way to access audio PCM data. By enabling event-based audio access for a specific channel via a SessionId, VaxVoIP triggers the OnCallSessionAccessAudioPCM() event, passing the PCM data to the application via the main thread. However, processing extensive tasks on the main thread may cause delays and impact the responsiveness of other events. This method is ideal for scenarios involving fewer calls and less intensive audio processing.

AI and Machine Learning Applications: By accessing raw PCM data, the VaxVoIP SIP Recording SDK enables AI-driven applications such as real-time speech recognition, language translation, and text-to-speech functionalities. This capability is crucial for applications that require real-time language translation of recorded VoIP calls or transcription services. AI models can process PCM audio to translate conversations between different languages, making the SDK ideal for compliance monitoring, legal transcription, and multilingual support in call recording solutions.

Please refer to the ACCESS AUDIO DATA PCM section for further details.

Syntax

```
boolean AccessAudioPCM(
                        SessionId,
                        PipeName,
                        AccessType
                        )
```

Parameters

SessionId (integer)
This parameter specifies a unique identifier for a call session.

PipeName (string)
The value of this parameter specifies the name of Pipe.

AccessType (integer)
The value of this parameter specifies the access type.

-1 = Access Type All None
0 = Access Type Event None
1 = Access Type Event Media Zero
2 = Access Type Event Media One
3 = Access Type Pipe None
4 = Access Type Pipe Media Zero
5 = Access Type Pipe Media One

Return Value

Upon successful execution, this function returns a non-zero value. If the execution fails, it returns zero. In the case of an error, a specific error code can be retrieved by calling the GetVaxErrorCode() method.

Example

```
OnIncomingCall(SessionId, CallerName, CallerId, CalleeName, CalleeId,
                    DialNo, FromPeerType, FromPeerName, UserAgentName,
                    RecXML, FromIP, FromPort)
{
        AccessAudioPCM(SessionId, "", 1)
        AcceptCallSession(SessionId, 20)
}

OnCallSessionAccessAudioPCM(SessionId, ChannelId, AccessType, DataPCM,
                                    SizePCM, TypePCM)
{
        // Access the DataPCM array and process it for audio analysis.
}
```

```
OnIncomingCall(SessionId, CallerName, CallerId, CalleeName, CalleeId,
                    DialNo, FromPeerType, FromPeerName, UserAgentName,
                    RecXML, FromIP, FromPort)
{
        PipeName = "\\\\.\\pipe\\PipeServer"+ ConvertToText(SessionId)

        AccessAudioPCM(SessionId, PipeName, 4)
        AcceptCallSession(SessionId, 20)

        // MSDN Microsoft Help: Create PipeClient and
        // Connect to PipeServer (PipeName)
}
```

See Also

OnCallSessionAccessAudioPCM(), SendAudioPCM()

RecordWaveSetCacheSize()

The RecordWaveSetCacheSize() function informs the VaxVoIP library (DLL) about the amount of memory to allocate for buffering recorded audio data before it is written to the storage file.

This setting helps optimize the performance of audio recording by ensuring that the library has sufficient memory to handle the data efficiently during the recording process.

Syntax

```
boolean RecordWaveStartToCallSession(CacheSize)
```

Parameters

CacheSize (integer)

This parameter specifies the number of bytes allocated for the cache. The default value is -1, which means there is no cache size limit. In this case, all recorded data is stored in memory (RAM) and is only written to the file when the call disconnects or the recording stops.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
RecordWaveSetCacheSize(1000000)  // 1MB cache size

OnCallSessionConnected(SessionId)
{
	RecordWaveStartToCallSession(SessionId, "Record.wav")
}
```

```
RecordWaveSetCacheSize(-1)  // No-limit cache size

OnCallSessionConnected(SessionId)
{
	RecordWaveStartToCallSession(SessionId, "Record.wav")
}
```

See Also

RecordWaveStartToCallSession(), RecordWaveStopToCallSession(), RecordWavePauseToCallSession(), GetVaxErrorCode()

RecordWaveStartToCallSession()

The RecordWaveStartToCallSession() function begins recording the audio stream of a Call-Session and saves it to a wave file (.wav).

If the FileName parameter is provided as an empty string, the RecordWaveStartToCallSession() function records the audio data directly into memory (RAM) instead of saving it as a file. When the call disconnects or the RecordWaveStopToCallSession() method is invoked, the recording stops. At this point, the OnCallSessionRecordedWaveREC() event is triggered, which provides the recorded wave file data as an array for further processing or analysis.

If the call disconnects or closes without explicitly stopping the recording using the RecordWaveStopToCallSession() method, VaxVoIP automatically stops the recording. It will then create a wave file using the specified FileName value, or if no FileName is provided, it will trigger the OnCallSessionRecordedWaveREC() event and pass the recorded wave file data as an array to the VaxVoIP-integrated application.

Syntax

```
boolean RecordWaveStartToCallSession(
                                        SessionId,
                                        FileName,
                                        TypeREC
                                        )
```

Parameters

SessionId (integer)
: This parameter specifies a unique identifier for a call session.

FileName (string)
: This parameter specifies the name of the file where the audio will be recorded. If an empty string is provided, the audio data is stored in memory (RAM) instead of being saved to a file.

TypeREC (integer)
: This parameter specifies the format and type of the recorded wave. The following values are supported:

  0 = REC TYPE MONO PCM
  1 = REC TYPE STEREO PCM
  2 = REC TYPE MONO G711U
  3 = REC TYPE STEREO G711U
  4 = REC TYPE MONO G711A
  5 = REC TYPE STEREO G711A

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
OnCallSessionConnected(SessionId)
{
	RecordWaveStartToCallSession(SessionId, "Record.wav", 0)
}

OnCallSessionClosed(SessionId, ReasonCode)
{

}
```

```
OnCallSessionConnected(SessionId)
{
	RecordWaveStartToCallSession(SessionId, "", 2)
}

OnCallSessionRecordedWaveREC(SessionId, DataREC, SizeREC, TypeREC)
{
	// DataREC represents a WAV file with header and audio data.
	// Save DataREC to a file with any desired name and .wav extension.
}
```

See Also

RecordWaveSetCacheSize(), OnCallSessionRecordedWaveREC(), RecordWaveStopToCallSession(), RecordWavePauseToCallSession(), GetVaxErrorCode()

RecordWaveStopToCallSession()

The RecordWaveStopToCallSession() function stops the recording of audio from a specific call-session. Although this function can be used to manually stop the recording, it is not necessary to use it to create the wave file. When a call ends or disconnects, VaxVoIP automatically completes and saves the recorded wave file, even if RecordWaveStopToCallSession() is not called.

Syntax

```
boolean RecordWaveStopToCallSession(SessionId)
```

Parameters

SessionId (integer)
This parameter specifies a unique identifier for a call session.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
OnCallSessionConnected(SessionId)
{
        RecordWaveStartToCallSession(SessionId, "Record.wav")
}
```

See Also

RecordWaveStartToCallSession(), RecordWavePauseToCallSession(), GetVaxErrorCode()

RecordWavePauseToCallSession()

The RecordWavePauseToCallSession() function pauses the recording of audio for a call-session. This allows for temporary suspension of the recording process, which can be resumed later or stopped completely. The paused recording is saved in memory and can be continued or finalized as needed

Syntax

```
boolean RecordWavePauseToCallSession(SessionId, Enable)
```

Parameters

SessionId (integer)
: This parameter specifies a unique identifier for a call session.

Enable (Boolean)
: This parameter specifies whether to pause or resume the recording process. Assign a value of 1 to pause the recording, or 0 to resume it.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
OnCallSessionConnected(SessionId)
{
        RecordWaveStartToCallSession(SessionId, "Record.wav")

        RecordWavePauseToCallSession(SessionId, 1)
}
```

See Also

RecordWaveStartToCallSession(), RecordWaveStopToCallSession(), GetVaxErrorCode()

AudioSessionLost()

The AudioSessionLost() method detects when an audio session is lost. VaxVoIP continuously monitors for incoming audio packets within a specified time interval. If no audio packets are received during this interval, VaxVoIP triggers the OnCallSessionLost() event, indicating that the audio session has been disrupted.

Syntax

```
boolean AudioSessionLost(Enable, Timeout)
```

Parameters

Enable (boolean)
: The value of this parameter can be 0 or 1. Assign value 1 to this parameter to enable voice session lost detection or 0 to disable it.

Timeout (integer)
: This parameter value specifies the time interval (seconds) for detection of voice data.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
AudioSessionLost(1, 20)

//Enabled audio session loss detection with a 20-second interval
```

See Also

OnCallSessionLost(), GetVaxErrorCode()

DiagnosticLogSIP()

The DiagnosticLogSIP() method provides the logging and monitoring of SIP messages.

VaxVoIP triggers OnIncomingDiagnosticLog()/OnOutgoingDiagnosticLog() event when it receives/sends SIP messages.

Syntax

```
boolean DiagnosticLogSIP(Inbound, Outbound)
```

Parameters

Inbound (boolean)
: The value of this parameter can be 0 or 1. To enable diagnostic of inbound voice stream assign value 1 to this parameter or 0 to disable it.

Outbound (boolean)
: The value of this parameter can be 0 or 1. To enable diagnostic of outbound voice stream assign value 1 to this parameter or 0 to disable it.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
DignosticLogSIP(1, 0)
```

See Also

OnOutgoingDiagnosticLog(), OnIncomingDiagnosticLog(), GetVaxErrorCode()

VaxServerStartTick()

The function VaxServerStartTick() sets a time interval, certain task is performed on expiry of this set interval.

When VaxServerStartTick() method is called, it sets a timer tick internally and trigger the tick event OnVaxServerTick() after that specific time.

Syntax

```
boolean VaxServerStartTick(TickId, Elapse)
```

Parameters

TickId (integer)
: This parameter specifies the unique tick identification.

Elapse (integer)
: The value of this parameter specifies the time (milliseconds).

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
/* Triggers OnVaxServerTick() after every 8 seconds */

VaxServerStartTick(2001, 8000)
```

See Also

VaxServerStopTick(), OnVaxServerTick(), GetVaxErrorCode()

VaxServerStopTick()

The VaxServerStopTick() method is used to stop the time counter for specified tick. It works just like KillTimer() win32 API.

Syntax

```
boolean VaxServerStopTick(TickId)
```

Parameters

TickId (integer)
This parameter specifies the unique tick identification

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
/* Triggers OnVaxServerTick() after every 8 seconds */

VaxServerStartTick(2001, 8000)

VaxServerStopTick(2001)
```

See Also

VaxServerStartTick(), OnVaxServerTick(), GetVaxErrorCode()

SetUserAgentName()

The SetUserAgentName() function sets the user-agent header field of SIP packet.

Syntax

```
boolean SetUserAgentName(Name)
```

Parameters

Name (string)
This parameter value specifies the User agent name.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
Result = SetUserAgentName("abc")
if(Result == 0) GetVaxErrorCode()
```

See Also

GetUserAgentName(), GetVaxErrorCode()

GetUserAgentName()

The GetUserAgentName() function returns the user-agent value previously set by calling SetUserAgentName() function.

Syntax

```
string GetUserAgentName()
```

Parameters

No parameters.

Return Value

The function returns the user agent name otherwise empty string.

Example

```
GetUserAgentName()
```

See Also

SetUserAgentName()

SetSessionNameSDP()

The SetSessionNameSDP() function sets the session-name field of SDP part of SIP packet.

Syntax

```
boolean SetSessionNameSDP(Name)
```

Parameters

Name (string)
: This parameter specifies the value that is to be set as session name of SIP packet.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
SetSessionNameSDP("xyz")
```

See Also

GetSessionNameSDP(), GetVaxErrorCode()

GetSessionNameSDP()

The GetSessionNameSDP() function returns the session-name value previously set by SetSessionNameSDP() function.

Syntax

```
string GetSessionNameSDP()
```

Parameters

No parameters.

Return Value

The function returns the session name otherwise empty string.

Example

```
GetSessionNameSDP()
```

See Also

SetSessionNameSDP()

GetCallSessionTxCodec()

The GetCallSessionTxCodec() returns audio codec that is being used by VaxVoIP to compress outbound voice stream of a specific Call-Session.

Syntax

```
integer GetCallSessionTxCodec(SessionId)
```

Parameters

SessionId (integer)
This parameter specifies a unique identifier for a call session.

Return Value

If the function succeeds, the return value is audio codec number.

If the function fails, the return value is -1. To get extended error information, call GetVaxErrorCode().

Audio Codec Numbers:

0 = GSM
1 = iLBC
2 = G711 A-Law
3 = G711 U-Law
4 = G729

Example

```
OnCallSessionConnected(SessionId)
{
	TxAudioCodec = GetCallSessionTxCodec(SessionId)
}
```

See Also

GetCallSessionRxCodec(), GetVaxErrorCode()

GetCallSessionRxCodec()

The GetCallSessionRxCodec() specifies audio codec that is being applied by VaxVoIP to inbound voice stream of a specific call-sesion.

Syntax

```
integer GetCallSessionRxCodec(SessionId)
```

Parameters

SessionId (integer)
This parameter specifies a unique identifier for a call session.

Return Value

If the function succeeds, the return value is audio codec number.

If the function fails, the return value is -1. To get extended error information, call GetVaxErrorCode().

Audio Codec Numbers:

0 = GSM
1 = iLBC
2 = G711 A-Law
3 = G711 U-Law
4 = G729

Example

```
OnCallSessionConnected(SessionId)
{
	RxAudioCodec = GetCallSessionRxCodec(SessionId)
}
```

See Also

GetCallSessionTxCodec(), GetVaxErrorCode()

CallSessionMuteVoice()

The CallSessionMuteVoice() mutes voice (listen or speak) of a specific call-session.

Syntax

```
boolean CallSessionMuteVoice(SessionId, Listen, Speak)
```

Parameters

SessionId (integer)
This parameter specifies a unique identifier for a call session.

Listen (boolean)
This parameter specifies to mutes the listening.

Speak (boolean)
This parameter specifies to mutes the speaking.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
OnIncomingCall(SessionId, CallerName, CallerId, CalleeName, CalleeId,
                DialNo, FromPeerType, FromPeerName, UserAgentName,
                RecXML, FromIP, FromPort)
{
    AcceptCallSession(SessionId, 20)

    AddCallSessionToConferenceRoom("TechRoom", SessionId)
    CallSessionMuteVoice(SessionId, 1, 0)
}
```

See Also

OnIncomingCall(), GetVaxErrorCode()

AddCustomHeader()

The AddCustomHeader() function can be used to add custom header fields in the SIP packets of different SIP requests.

Some of the SIP requests; REGISTER, INVITE, ACK, CANCEL, BYE, OPTIONS

Syntax

```
boolean AddCustomHeader(ReqId, Name, Value)
```

Parameters

ReqId (integer)
: This parameter specifies a unique identification of a SIP request. Supported ReqId values are;

0 = INVITE
1 = REFER
2 = CANCEL
3 = BYE

Name (string)
: This parameter specifies the name of custom header field.

Value (string)
: This parameter specifies the value of custom header field.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
Result = Initialize("sipsdk.com")
if(Result = 0) GetVaxErrorCode()

AddCustomHeader(0, "Call_Info", "WaitingTime = 0")
```

See Also

RemoveCustomHeader(), RemoveCustomHeaderAll(),

RemoveCustomHeader()

The RemoveCustomHeader() function removes the custom header fields added by using AddCustomHeader() function.

Syntax

```
boolean RemoveCustomHeader(ReqId,  Name)
```

Parameters

ReqId (integer)
: This parameter specifies a unique identification of a SIP request. Supported ReqId values are;

0 = INVITE
1 = REFER
2 = CANCEL
3 = BYE

Name (string)
: This parameter specifies the custom header field.

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
RemoveCustomHeader(0, "Call_Info")
```

See Also

AddCustomHeader(), RemoveCustomHeaderAll()

RemoveCustomHeaderAll()

The RemoveCustomHeaderAll() function removes all custom header fields added by using AddCustomHeader() function.

Syntax

```
boolean RemoveCustomHeaderAll(ReqId)
```

Parameters

ReqId (integer)
This parameter specifies a unique identification of a SIP request. Supported ReqId values are;

0 = INVITE
1 = REFER
2 = CANCEL
3 = BYE

Return Value

On successful execution, this function returns a non-zero value. If it fails, it returns 0. The specific error code can be retrieved by using the GetVaxErrorCode() method.

Example

```
RemoveCustomHeaderAll(0)
```

See Also

AddCustomHeader(), RemoveCustomHeader()

GetCallSessionHeaderCallId()

The GetCallSessionHeaderCallId() returns the unique field/header CallId value for the specifc call session. This is actually a SIP packet unique CallId.

Syntax

```
string GetCallSessionHeaderCallId (SessionId)
```

Parameters

SessionId (integer)
This parameter specifies the unique identification of a call.

Return Value

If the function succeeds, the return value is header call-id field otherwise, it returns empty string. To get extended error information, call GetVaxErrorCode().

Example

```
OnIncomingCall(SessionId, CallerName, CallerId, CalleeName, CalleeId,
                DialNo, FromPeerType, FromPeerName, UserAgentName,
                RecXML, FromIP, FromPort)
{
    HeaderId = GetCallSessionHeaderCallId(SessionId)
}
```

See Also

OnCallSessionCreated()

AttackDetectScanSIP()

The AttackDetectScanSIP() method strengthens the security of a VaxVoIP SDK-based SIP server by detecting and preventing unauthorized SIP traffic. This method accepts a domain name as a parameter and monitors incoming SIP packets for that domain by inspecting the FromURI field.

If the domain name provided as a parameter does not match the domain in the FromURI field of an incoming SIP packet, the server identifies the packet as part of a potential attack and triggers the **OnAttackDetectedScanSIP()** event. In response, the SIP server can block the attacker's IP and port using the Windows Defender Firewall APIs.

To mitigate risks, the server discards such packets without sending any SIP response, effectively neutralizing the threat without engaging the attacker.

This mechanism ensures that only legitimate SIP requests associated with the specified domain are processed, reducing the risk of malicious activities such as unauthorized port scanning or domain spoofing. By silently discarding invalid packets, the server also minimizes its exposure to further probing attempts.

Syntax

```
boolean AttackDetectScanSIP(DomainName)
```

Parameters

DomainName (string)
: This parameter specifies the domain name.

Return Value

On successful execution, this function returns a non-zero value. If the function fails, it returns 0. To determine the specific error, use the GetVaxErrorCode() method.

Example

```
AttackDetectScanSIP ("demo.vaxvoip.com")
```

On the softphone or SIP client side, configure the domain "demo.vaxvoip.com" as the DomainRealm. The SIP client will then include this domain in its SIP URIs.

See Also

AttackDetectFloodSIP(), AttackDetectBruteForceSIP(), OnAttackDetectedScanSIP()

AttackDetectFloodSIP()

The AttackDetectFloodSIP() method monitors the number of incoming SIP requests per second to detect potential flooding attacks. If the number of SIP requests per second exceeds the threshold specified as a parameter, the method identifies it as a threat and triggers the **OnAttackDetectedFloodSIP()** event.

Upon detecting such an attack, the SIP server, built using the VaxVoIP SIP SDK, can block the attacker's IP address and port by utilizing the Windows Defender Firewall APIs. This automatic blocking mechanism helps prevent further intrusion or disruption caused by the flood of requests.

By implementing this method, the SIP server enhances its resilience against Denial-of-Service (DoS) attacks and ensures smooth communication for legitimate traffic. The configurable threshold allows administrators to fine-tune the server's sensitivity to flooding attempts based on their specific requirements.

Syntax

```
boolean AttackDetectFloodSIP(ReqRecvLimit)
```

Parameters

ReqRecvLimit (integer)
: This parameter specifies the threshold for the maximum number of incoming SIP requests allowed per second. If the number of requests exceeds this limit, the method triggers an attack detection event, signaling a potential flood attack.

Return Value

On successful execution, this function returns a non-zero value. If the function fails, it returns 0. To determine the specific error, use the GetVaxErrorCode() method.

Example

```
Initialize("")

AttackDetectFloodSIP(1000)
```

See Also

AttackDetectScanSIP(), AttackDetectBruteForceSIP(), OnAttackDetectedFloodSIP()

AttackDetectBruteForceSIP()

The AttackDetectBruteForceSIP() method monitors the authentication failure attempts for a specific interval of time. If attempts increase more than the value provided as first parameter then VaxVoIP SDK detects it as threat and informs application by triggering event **OnAttackDetectedBruteForceSIP()**, application can block the IP and port by using Microsoft's provided Firewall Defender APIs.

Syntax

```
boolean AttackDetectBruteForceSIP(FailureCount, FailureInterval)
```

Parameters

FailureCount (integer)
: This parameter specifies the maximum number of authentication failure attempts allowed within the time interval defined by the FailureInterval. If the number of failed attempts exceeds this limit, the method will trigger the brute force attack detection.

FailureInterval: (integer)
: This parameter defines the time interval (in seconds) within which the authentication failure attempts are counted. If the number of failed attempts within this interval exceeds the FailureCount, the system detects a potential brute force attack.

Return Value

On successful execution, this function returns a non-zero value. If the function fails, it returns 0. To determine the specific error, use the GetVaxErrorCode() method.

Example

```
Initialize("")

AttackDetectBruteForceSIP(10, 4)
```

See Also

AttackDetectScanSIP(), AttackDetectFloodSIP(), OnAttackDetectedBruteForceSIP()

EXPORTED EVENTS

OnVaxErrorLog()

VaxVoIP triggers OnVaxErrorLog() when execution of any function fails.

Please see LIST OF ERROR CODES for more details.

Syntax

```
OnVaxErrorLog(FuncName, ErrorCode, ErrorMsg)
```

Parameters

FuncName (string)
This parameter value specifies name of the function.

ErrorCode (integer)
This parameter value specifies error code.

ErrorMsg (string)
This parameter value specifies error text message.

Example

```
Result = Initialize("")

// On failure of Initialize(), the OnVaxErrorLog() event is triggered

if (Result = false)
{
	GetVaxErrorCode()
	return
}

OnVaxErrorLog(FuncName, ErrorCode, ErrorMsg)
{
	//Log or store the method name provided in FuncName
	//Log or store the error message provided in ErrorMsg
}
```

See Also

GetVaxErrorCode(), OnCallSessionErrorLog()

OnCallSessionErrorLog()

OnCallSessionErrorLog() is an event that notifies the application when an internal function execution fails. It provides detailed information about errors encountered during call session functions, which helps in diagnosing and troubleshooting issues effectively.

Please see **LIST OF ERROR CODES** for more details.

Syntax

```
OnCallSessionErrorLog(SessionId, ChannelId, ErrorCode, ErrorMsg)
```

Parameters

SessionId (integer)
: This parameter specifies a unique identifier for a call session.

ChannelId (integer)
: This parameter identifies a call within a call session.

  0 = Channel-ZERO call
  1 = Channel-ONE call

ErrorCode (integer)
: Specifies the error code associated with the failure.

ErrorMsg (string)
: Provides a textual description of the error message.

Example

```
OnCallSessionErrorLog(SessionId, ChannelId, ErrorCode, ErrorMsg)
{
	//Log or store the error message provided in ErrorMsg
}
```

See Also

GetVaxErrorCode(), OnVaxErrorLog()

OnCallSessionCreated()

The OnCallSessionCreated() event triggers when VaxVoIP creates/allocates a call-session internally.

Syntax

```
OnCallSessionCreated(
                        SessionId,
                        ReasonCode
                        )
```

Parameters

SessionId (integer)
: This parameter specifies the unique identification of a call-session.

ReasonCode(integer)
: This parameter specifies the reason due to which the call-session is created.

- INCOMING_CALL 1001
- OUTGOING_CALL 1002
- MERGED 1003
- TRANSFERED 1004

Example

```
OnCallSessionCreated(SessionId, ReasonCode)
{
        // Allocate resources for the Session
}
```

See Also

OnCallSessionClosed()

OnCallSessionClosed()

The OnCallSessionClosed() event triggers when VaxVoIP closes a call-session internally.

Syntax

```
OnCallSessionClosed(SessionId, ChannelId, ReasonCode)
```

Parameters

SessionId (integer)
This parameter specifies a unique identifier for a call session.

ChannelId (integer)
This parameter identifies a call within a call session.

0 = Channel-ZERO call
1 = Channel-ONE call

ReasonCode (integer)
This parameter specifies the reason due to which the call-session is closed.

- HANGUP 0
- SESSION_LOST 1
- MERGED 2
- TRANSFERED 3
- REJECTED 4
- FAILED 5
- CANCELLED 6
- TIMEOUT 7
- CLOSED 8

Example

```
OnCallSessionClosed(SessionId, ChannelId, ReasonCode)
{

}
```

See Also

OnCallSessionCreated()

OnCallSessionAccessAudioPCM()

The OnCallSessionAccessAudioPCM() event is triggered when event-based access to the audio PCM data in real-time of a call within a call session is activated using the AccessAudioPCM() method.

This event-based approach provides a simplified and efficient way to access real-time audio PCM data. When event-based audio access is enabled for a specific channel associated with a SessionId, VaxVoIP automatically triggers the OnCallSessionAccessAudioPCM() event.

During this event, the PCM audio data is passed directly to the application, allowing developers to process or manipulate the audio stream as needed.

This mechanism is particularly useful for applications requiring real-time audio analysis, recording, or other custom audio processing tasks within the VaxVoIP framework.

Syntax

```
OnCallSessionAccessAudioPCM(
                                SessionId,
                                AccessType,
                                DataPCM,
                                SizePCM,
                                TypePCM
                                )
```

Parameters

SessionId (integer)
: This parameter specifies a unique identifier for a call session.

AccessType (integer)
: The value of this parameter specifies the access type.

DataPCM (array)
: Contains the raw audio PCM data from the call session.

SizePCM (integer)
: Specifies the size of the PCM data in bytes.

TypePCM (integer)
: Reserved and set to 0 for future use.

Example

```
OnCallSessionAccessAudioPCM(SessionId, ChannelId, AccessType, DataPCM,
                                    SizePCM, TypePCM)
{
        // Access the DataPCM array and process it for audio analysis.
}
```

See Also

AccessAudioPCM()

OnRegisterUser()

The OnRegisterUser() event is triggered whenever VaxVoIP receives a registration request from any SIP client. This event occurs as part of the SIP registration process, where the SIP client attempts to register its information with the SIP server.

Handling this event allows the system to manage and process incoming registration requests, ensuring that SIP clients are properly authenticated and registered within the VaxVoIP environment

Please see SIP CLIENT REGISTRATION PROCESS for more details.

Syntax

```
OnRegisterUser(
        UserName,
        Domain,
        UserAgentName,
        FromIP,
        FromPort,
        RegId
        )
```

Parameters

UserName (string)
: This parameter specifies the user's login of SIP client.

Domain (string)
: This parameter specifies the domain and its value is used to configure and register the SIP clients to VaxVoIP and other SIP servers.

UserAgentName (string)
: This parameter specifies the UserAgentName of SIP client.

FromIP (string)
: This parameter value specifies the from IP address.

FromPort (integer)
: This parameter specifies the from port number.

RegId (integer)
: This parameter specifies a unique identification of a registration session.

Example

```
OnRegisterUser(UserName, Domain, UserAgentName, FromIP, FromPort,
                    RegId)
{
        AddUser(UserName, "123", "01")
        AcceptRegister(RegId)
}
```

See Also

OnUnRegisterUser(), AddUser(), RemoveUser()

OnRegisterUserSuccess()

The OnRegisterUserSuccess() event triggers when SIP client successfully registers to VaxVoIP server.

Please see SIP CLIENT REGISTRATION PROCESS for more details.

Syntax

```
OnRegisterUserSuccess(UserName)
```

Parameters

UserName (string)
: This parameter specifies the user's login of SIP client.

FromIP (string)
: This parameter value specifies the from IP address.

FromPort (integer)
: This parameter specifies the from port number.

RegId (integer)
: This parameter specifies a unique identification of a registration session.

Example

```
OnRegisterUserSuccess(UserName, FromIP, FromPort, RegId)
{
}
```

See Also

OnRegisterUser(), OnRegisterUserFailed()

OnRegisterUserFailed()

The OnRegisterUserFailed() event triggers when SIP client fails to register with VaxVoIP server.

Syntax

```
OnRegisterUserFailed(UserName)
```

Parameters

UserName (string)
: This parameter specifies the user's login of SIP client.

FromIP (string)
: This parameter value specifies the from IP address.

FromPort (integer)
: This parameter specifies the from port number.

RegId (integer)
: This parameter specifies a unique identification of a registration session.

Example

```
OnRegisterUserFailed(UserName, FromIP, FromPort, RegId)
{
	RemoveUser(UserName)
}
```

See Also

OnRegisterUser(), OnRegisterUserSuccess(), RemoveUser()

OnUnRegisterUser()

VaxVoIP triggers the OnUnRegisterUser() event when it receives an unregister request from any SIP client.

This event occurs when a SIP client requests to deregister from the SIP server, allowing the system to handle and process the removal of the client's registration information from the VaxVoIP environment.

Please see SIP CLIENT REGISTRATION PROCESS for more details.

Syntax

```
OnUnRegisterUser(UserName)
```

Parameters

UserName (string)
: This parameter specifies the user's login of SIP client.

Example

```
OnUnRegisterUser(UserName)
{
        RemoveUser(UserName)
}
```

See Also

OnRegisterUser(), RemoveUser(), AddUser()

OnLineRegisterTrying()

VaxVoIP triggers OnLineRegisterTrying() event when it receives SIP response "100, Trying" from other SIP server.

VaxVoIP connect and work with other external SIP servers and IP-Telephony Service providers by using AddLine() and RegisterLine() functions.

Please see **HOW TO CONNECT TO IP-TELEPHONY SERVICE PROVIDER (ITSP)** for more details.

Please see **HOW TO CONNECT TO PSTN/GSM NETWORK** for more details.

Syntax

```
OnLineRegisterTrying(LineName)
```

Parameters

LineName (string)
: This parameter value specifies the unique line name to identify a specific line.

Example

```
OnLineRegisterTrying(LineName)
{
}
```

See Also

RegisterLine(), AddLine(), OnLineRegisterFailed(), OnLineRegisterSuccess()

OnLineRegisterFailed()

VaxVoIP triggers OnLineRegisterFailed() event when registration of a LINE (SIP account settings) to external SIP server or IP-Telephony service provider fails.

Syntax

```
OnLineRegisterFailed(
                        LineName,
                        StatusCode,
                        ReasonPhrase
                        )
```

Parameters

LineName (string)
: This parameter value specifies the unique line name to identify a specific line.

StatusCode (integer)
: This parameter specifies SIP response status code (408, 403 etc). **LIST OF SIP RESPONSES**

ReasonPhrase (string)
: This parameter specifies SIP response reason phrase (Request Timeout, Forbidden etc).

Example

```
OnLineRegisterFailed(LineName, StatusCode, ReasonPhrase)
{
}
```

See Also

AddLine(), RegisterLine(), OnLineRegisterTrying(), OnLineRegisterSuccess()

OnLineRegisterSuccess()

VaxVoIP triggers OnLineRegisterSuccess() event when line (SIP account settings) registration request (by RegisterLine() function) to external SIP server or IP-Telephony service provider successfully completes.

Please see **HOW TO CONNECT TO IP-TELEPHONY SERVICE PROVIDER (ITSP)** for more details.

Please see **HOW TO CONNECT TO PSTN/GSM NETWORK** for more details.

Syntax

```
OnLineRegisterSuccess(LineName)
```

Parameters

LineName (string)
: This parameter value specifies the unique line name to identify a specific line.

Example

```
OnLineRegisterSuccess(LineName)
{
}
```

See Also

AddLine(), RegisterLine(), OnLineRegisterTrying(), OnLineRegisterFailed()

OnLineUnRegisterTrying()

VaxVoIP triggers OnLineUnRegisterTrying() event when it receives SIP response "100, Trying" from other SIP server during unregister process.

To unregister or disconnect VaxVoIP from external third party SIP Server the UnRegisterLine() is used.

Syntax

```
OnLineUnRegisterTrying(LineName)
```

Parameters

LineName (integer)
: This parameter value specifies the unique line name to identify a specific line.

Example

```
OnLineUnRegisterTrying(LineName)
{
}
```

See Also

RegisterLine(), UnRegisterLine(), AddLine(), OnLineUnRegisterFailed(), OnLineUnRegisterSuccess()

OnLineUnRegisterFailed()

VaxVoIP triggers OnLineUnRegisterFailed() event if a provided LINE fails to un-register from external SIP server or IP-Telephony service provider.

Syntax

```
OnLineUnRegisterFailed(
                        LineName,
                        StatusCode,
                        ReasonPhrase
                        )
```

Parameters

LineName (string)
: This parameter value specifies the unique line name to identify a specific line.

StatusCode (integer)
: This parameter specifies SIP response status code (408, 403 etc). **LIST OF SIP RESPONSES**

ReasonPhrase (string)
: This parameter specifies SIP response reason phrase (Request Timeout, Forbidden etc).

Example

```
OnLineUnRegisterFailed(LineName)
{
}
```

See Also

RegisterLine(), UnRegisterLine(), AddLine(), OnLineUnRegisterTrying(), OnLineUnRegisterSuccess()

OnLineUnRegisterSuccess()

VaxVoIP triggers OnLineUnRegisterSuccess() event when line unregisters successfully from external SIP server or IP-Telephony service provider.

VaxVoIP calls UnRegisterLine() function to un-register/disconnect a line (SIP account settings) from external SIP server or IP-Telephony service provider and if unregister request is successfully executes then OnLineUnRegisterSuccess() event triggers.

Syntax

```
OnLineUnRegisterSuccess(LineName)
```

Parameters

LineName (string)
: This parameter value specifies the unique line name to identify a specific line.

Example

```
OnLineUnRegisterSuccess(LineName)
{
}
```

See Also

RegisterLine(), UnRegisterLine(), AddLine(), OnLineUnRegisterTrying(), OnLineUnRegisterFailed()

OnIncomingCall()

The OnIncomingCall() event triggers when VaxREC receives a SIP-REC based call request.

Syntax

```
OnIncomingCall(SessionId, CallerName, CallerId, CalleeName, CalleeId,
               DialNo, FromPeerType, FromPeerName, UserAgentName,
               RecXML, FromIP, FromPort)
```

Parameters

SessionId (integer)
This parameter specifies a unique identification of a Call-Session.

CallerName (string)
This parameter specifies the caller name.

CallerId (string)
This parameter specifies a unique identification of caller.

CalleeName (string)
This parameter specifies the callee name.

CalleeId (string)
This parameter specifies a unique identification of callee.

DialNo (string)
This parameter value specifies the number to be dialed.

FromPeerType (integer)
This parameter value specifies the type of From-Peer.

0 = User PeerType
1 = Line PeerType

FromPeerName (string)
This parameter value specifies the name of From-Peer.

UserAgentName (string)
This parameter value specifies the name of UserAgent.

RecXML (string)
This parameter specifies a XML part of SIP/SDP packet.

FromIP (string)
This parameter value specifies the From IP address.

FromPort (integer)
This parameter specifies the From port number.

Example

```
OnIncomingCall(SessionId, CallerName, CallerId, CalleeName, CalleeId,
                DialNo, FromPeerType, FromPeerName, UserAgentName,
                RecXML, FromIP, FromPort)
{

}
```

See Also

AcceptCallSession(), OnCallSessionConnected, OnCallSessionFailed()

OnCallSessionFailed()

The OnCallSessionFailed() event triggers when VaxVoIP receives failure responses during call connection process and failed to established a Call-Session with SIP client/third party server.

Syntax

```
OnCallSessionFailed(
                    SessionId,
                    ChannelId,
                    StatusCode,
                    ReasonPhrase
                    )
```

Parameters

SessionId (integer)
This parameter specifies a unique identifier for a call session.

ChannelId (integer)
This parameter identifies a call within a call session.

0 = Channel-ZERO call
1 = Channel-ONE call

StatusCode (integer)
This parameter specifies SIP response status code (486, 404 etc).
LIST OF SIP RESPONSES

ReasonPhrase (string)
This parameter specifies SIP response reason phrase (Busy here, Not found etc).

Example

```
OnCallSessionFailed(SessionId, ChannelId, StatusCode, ReasonPhrase)
{
}
```

See Also

AcceptCallSession(), SplitCallSession(), OnCallSessionConnected, OnIncomingCall()

OnCallSessionConnected()

The OnCallSessionConnected() event triggers when VaxREC successfully established a Call-Session with SIP-REC client or remote SIP-REC SIP Server.

Syntax

```
OnCallSessionConnected(SessionId)
```

Parameters

SessionId (integer)
This parameter specifies a unique identification of a Call-Session.

Example

```
OnCallSessionConnected(SessionId)
{
	RecordWaveStartToCallSession(SessionId, "RecordCall.wav", 1)
}
```

Example

```
OnCallSessionConnected(SessionId)
{
	RecordWaveStartToCallSession(SessionId, "")
}

OnCallSessionRecordedWaveREC(SessionId, DataREC, SizeREC)
{
	//Create a file, and write data or process data
}
```

See Also

OnCallSessionRecordedWaveREC(), AcceptCallSession(), OnIncomingCall()

OnCallSessionLost()

The OnCallSessionLost() event triggers when VaxREC does not receive audio media packets for define interval of time.

Syntax

```
OnCallSessionLost(SessionId)
```

Parameters

SessionId (integer)
This parameter specifies a unique identification of a Call-Session.

Example

```
OnCallSessionLost(SessionId)
{
}
```

See Also

AudioSessionLost()

OnCallSessionHangup()

The OnCallSessionHangup() event triggers when remote party hangup the call.

Syntax

```
OnCallSessionHangup(SessionId)
```

Parameters

SessionId (integer)
This parameter specifies a unique identification of a Call-Session.

Example

```
OnCallSessionHangup(SessionId)
{
}
```

See Also

AcceptCallSession(), OnCallSessionConnected(), OnIncomingCall()

OnCallSessionTimeout()

The OnCallSessionTimeout() event triggers when VaxREC fails to establish a Call-Session and does not receive a response from the SIP-REC client within the specified time period, as set in the AcceptCallSession() method. This event serves as a notification that the Call-Session establishment process has timed out, allowing for appropriate handling in the application.

Syntax

```
OnCallSessionTimeout(SessionId)
```

Parameters

SessionId (integer)
This parameter specifies a unique identification of a Call-Session.

Example

```
OnCallSessionTimeout(SessionId)
{
}
```

See Also

AcceptCallSession()

OnCallSessionCancelled()

The `OnCallSessionCancelled()` event triggers when the caller cancels the call before it is accepted by VaxREC. This event provides notification that a call session has been canceled by the caller, allowing the application to handle this event accordingly.

Syntax

```
OnCallSessionCancelled(SessionId)
```

Parameters

SessionId (integer)
: This parameter specifies a unique identification of a Call-Session.

Example

```
OnCallSessionCancelled(SessionId)
{
}
```

See Also

OnOutgoingDiagnosticLog()

The OnOutgoingDiagnosticLog() event triggers when VaxREC sends a SIP packet. This event can be used for logging and monitoring of outbound SIP messages.

Syntax

```
OnOutgoingDiagnosticLog(
                        MsgSIP,
                        ToIP,
                        ToPort
                        )
```

Parameters

MsgSIP (string)
: This parameter value specifies the SIP packet message.

ToIP (string)
: This parameter value specifies the To IP address.

ToPort (integer)
: This parameter value specifies the To port number.

Example

```
OnOutgoingDiagnosticLog(MsgSIP, ToIP, ToPort)
{

}
```

See Also

DiagnosticLogSIP(), OnIncomingDiagnosticLog()

OnIncomingDiagnosticLog()

The OnIncomingDiagnosticLog() event triggers when VaxREC receives a SIP packet. This event can be used for logging and monitoring of inbound SIP messages.

Syntax

```
OnIncomingDiagnosticLog(
                        MsgSIP,
                        FromIP,
                        FromPort
                        )
```

Parameters

MsgSIP (string)
: This parameter value specifies the SIP packet message.

FromIP (string)
: This parameter value specifies the From IP address.

FromPort (integer)
: This parameter specifies the From port number.

Example

```
OnIncomingDiagnosticLog(MsgSIP, FromIP, FromPort)
{

}
```

See Also

DiagnosticLogSIP(), OnOutgoingDiagnosticLog()

OnVaxServerTick()

The OnVaxServerTick() event triggers after a specified time interval set by VaxServerStartTick() function.

Syntax

```
OnVaxServerTick(TickId)
```

Parameters

TickId (integer)
This parameter specifies the unique tick identification.

Example

```
OnVaxServerTick(TickId)
{
}
```

See Also

VaxServerStartTick(), VaxServerStopTick()

OnCallSessionRecordedWaveREC()

The OnCallSessionRecordedWaveREC() event is triggered when a recorded wave file of a specific call is available. This event provides the recorded audio data in the form of an array, which represents the wave file.

To initiate recording, use the RecordWaveStartToCallSession() method with an empty string for the FileName parameter. This method begins recording and stores the audio data directly in memory (RAM).

When the recording needs to be stopped, either due to the call being disconnected or by invoking the RecordWaveStopToCallSession() method, the recording is halted. Subsequently, the OnCallSessionRecordedWaveREC() event is triggered, delivering the recorded wave file data as an array for further processing or analysis.

Syntax

```
OnCallSessionRecordedWaveREC(
                                    SessionId,
                                    DataREC,
                                    SizeREC,
                                    TypeREC
                                     )
```

Parameters

SessionId (integer)
: This parameter specifies a unique identifier for a call session.

DataREC (array)
: This parameter specifies the complete wave file.

SizeREC (integer)
: This parameter specifies the size of wave file.

TypeREC (integer)
: The parameter specifies the type of wave file.

REC TYPE MONO PCM	0
REC TYPE STEREO PCM	1
REC TYPE MONO G711U	2
REC TYPE STEREO G711U	3
REC TYPE MONO G711A	4
REC TYPE STEREO G711A	5

Example

```
OnCallSessionConnected(SessionId)
{
        RecordWaveStartToCallSession(SessionId, “”, 2)
}

OnCallSessionRecordedWaveREC(SessionId, DataREC, SizeREC, TypeREC)
{
        // DataREC is a WAV file containing the header & audio data.
        // Write the DataREC to a file with any name and a .wav extension.
}
```

See Also

RecordWaveStartToCallSession()

OnAttackDetectedScanSIP()

The OnAttackDetectedScanSIP() event is triggered when the SIP server detects a potential attack, such as unauthorized or suspicious SIP traffic, based on certain criteria defined in the **AttackDetectScanSIP()** method.

Syntax

```
OnAttackDetectedScanSIP(
                        ReqMethod,
                        AddrIP,
                        AddrPort,
                        AddrType
                        )
```

Parameters

ReqMethod (string)
: Represents the request method in the incoming SIP packet, such as "INVITE," "REGISTER," or other SIP methods. This helps identify the type of SIP request that triggered the detection event.

AddrIP (string)
: The IP address of the source that sent the suspicious SIP request. This allows identification of the source and can be used for blocking or investigating the IP further.

AddrPort (integer)
: The port number of the source address from which the SIP request was sent. This helps pinpoint the exact source and enables blocking or restricting traffic from that port.

AddrType (integer)
: Specifies the transport protocol used for the SIP request.

  0: UDP (User Datagram Protocol)
  1: TCP (Transmission Control Protocol)
  2: TLS (Transport Layer Security)

  This parameter indicates the transport protocol type, which is useful for differentiating between network communication types.

Example

```
AttackDetectScanSIP("demo.sipsdk.com")

OnAttackDetectedScanSIP(ReqMethod, AddrIP, AddrPort, AddrType)
{
        // Use Windows Defender Firewall APIs
        // Run command prompt-based Firewall commands
        // Block AddrIP and AddrPort.
}
```

See Also

AttackDetectScanSIP()

OnAttackDetectedFloodSIP()

The OnAttackDetectedFloodSIP() event is triggered when the SIP server detects a potential flooding attack, which occurs when the number of incoming SIP requests exceeds the threshold defined by the **AttackDetectFloodSIP()** method. This event provides details about the source of the flooding attempt, such as the IP address, port, and protocol type used by the attacker.

Syntax

```
OnAttackDetectedFloodSIP(
                        AddrIP,
                        AddrPort,
                        AddrType
                        )
```

Parameters

AddrIP (string)
: This parameter contains the IP address of the source that sent the excessive SIP requests. It identifies the origin of the flood and can be used for blocking or further investigation.

AddrPort (integer)
: This parameter specifies the port number from which the flood of SIP requests was sent. It helps pinpoint the source of the attack at the network level, allowing the server to block or restrict traffic from that port.

AddrType (integer)
: Indicates the transport protocol used for the SIP requests.

  0: UDP (User Datagram Protocol)
  1: TCP (Transmission Control Protocol)
  2: TLS (Transport Layer Security)

  This parameter helps identify the transport protocol used by the attacker, which can be useful for applying specific security measures based on the protocol type.

Example

```
AttackDetectFloodSIP(2000)

AttackDetectFloodSIP(AddrIP, AddrPort, AddrType)
{
	// Use Windows Defender Firewall APIs
	// Run command prompt-based Firewall commands
	// Block AddrIP and AddrPort.
}
```

See Also

AttackDetectFloodSIP()

OnAttackDetectedBruteForceSIP()

The OnAttackDetectedBruteForceSIP() event is triggered when the SIP server detects a potential brute-force attack, which occurs when there are multiple failed authentication attempts within a specified time period. This event provides detailed information about the failed attempts, including the request method, the number of failed attempts, and the source of the attack.

Syntax

```
OnAttackDetectedFloodSIP(ReqMethod, AuthFailureCount, AddrIP,
                                AddrPort, AddrType)
```

Parameters

ReqMethod (string)
: Represents the SIP request method (e.g., "INVITE," "REGISTER") associated with the authentication failures. This helps identify which SIP method the brute-force attempts are targeting.

AuthFailureCount (integer)
: The number of authentication failure attempts that were detected within the time window specified by the **AttackDetectBruteForceSIP()** method. If this count exceeds the defined threshold, it triggers the event to alert about the potential brute-force attack.

AddrIP (string)
: The IP address of the source from which the failed authentication attempts originated. This helps to identify the source of the attack and can be used for blocking or further investigation.

AddrPort (integer)
: The port number from which the failed authentication attempts were made. This helps pinpoint the exact source port for further action or investigation.

AddrType (integer)
: Indicates the transport protocol used for the SIP requests.

  0: UDP (User Datagram Protocol)
  1: TCP (Transmission Control Protocol)
  2: TLS (Transport Layer Security)

  This parameter helps identify the transport protocol used by the attacker, which can be useful for applying specific security measures based on the protocol type.

Example

```
AttackDetectBruteForceSIP(200, 5)  // 5 seconds

OnAttackDetectedFloodSIP(ReqMethod, AuthFailureCount, AddrIP,
                         AddrPort, AddrType)
{
        // Use Windows Defender Firewall APIs
        // Run command prompt-based Firewall commands
        // Block AddrIP and AddrPort.
}
```

See Also

AttackDetectBruteForceSIP()

LIST OF ERROR CODES

200	Failed to initialize RTP Socket.
201	Failed to allocate RTP port or provided value of RTP listen IP is incorrect.
202	Failed to create RTP task manager.
203	Failed to start media thread.
204	Unable to create RTP communication manager.
205	Unable to run RTP communication manager.
206	Unable/failed to open file.
207	Unable to read file.
208	Incorrect wave file format, please use 8000Hz, 16bit, mono uncompressed wave file.
209	Provided wave id is not valid.
210	Unable to start recording manager.
211	Voice session is not connected or started yet.
212	Failed to initialize VaxVoIP Library.
213	Unable to construct SIP SDP body.
214	Unable to construct SIP INVITE request.
215	Error to initialize SIP communication layer.
216	Failed to open SIP listen port or provided SIP listen IP is incorrect.
217	Failed to create SIP task manager.
218	Unable to create SIP REGISTER request.
219	Unable to create SIP UN-REGISTER request.
220	Unable to create SIP BYE request.
221	Unable to create SIP CANCEL request.
222	Provided SessionId does not exist.
223	Voice session does not exist.
224	Provided login does not exist.
225	Login length is incorrect.
226	Provided line does not exist.
227	Can't send SIP response.
228	Invalid SIP response code, please check SIP RFC 3261.
229	Error to create SIP response.
230	Provided line already exist.
231	Incorrect RegId or RegId does not exist.
232	Error to create SIP response.
233	User is not registered.
234	Invalid codec OR no codec found for voice streaming.
235	Invalid DTMF type.
236	Remote party does not support RFC2833 DTMF digits.
237	Error to create REFER request to transfer the call.
238	Error to create event handler.
239	License key is not valid or expired.
240	Invalid digit for DTMF.
241	Invalid proxy URI.
242	Line is not registered.
243	Invalid SIP To-URI.

244	Desired operation can only be performed on connected session.
245	Can't perform desired operation on connected session.
246	Direct communication is not supported.
247	One of the provided parameter(s) value is not correct.
248	Invalid chat message-Id.
249	Invalid subscribe-Id.
250	Failed to generate a unique-Id.
251	Provided unique-Id is not valid.
252	Provided unique-Id or value already exist.
253	Provided unique-Id or value does not exist.
254	Failed to create session.
255	Failed to enable DTMF detection.
256	Error to process transfer request.
257	Provided ListenIP is already binded.
258	Provided ListenIP does not exist in Server Key.
259	Provided SessionId has No Free Channel.
260	Both users (pickup and ringing) should be members of same pickup group.
261	Crypto audio or video media mismatched.
262	Unable to create socket dispatcher.
263	Unable to post message to socket dispatcher.
264	Provided addr is not valid.
265	Unable to bind socket addr or IP.
266	Failed to allocate socket port or provided value of listen IP or port is already in use.
267	General socket or network failure.
268	Failed to add wait-event to socket dispatcher.
269	Unable to create socket (timeout).
270	Line catch-all can't be used.
271	Route prefix conflicts with another route.
272	Unable to find provided SessionId.
273	Unable to find incoming call or inbound call does not exist.
274	Inbound call already connected.
275	Unable to connect to the destination address. Provided address or port is not valid.
276	Unable to capture or bind to the destination address. Provided address or port is not valid.
277	User already attached.
278	Unable to create thread dispatcher.
279	Unable to post message to thread dispatcher.
280	Unable to process to thread dispatcher.
281	Terminating abnormally thread dispatcher.
282	Unable to create pipe dispatcher.
283	Unable to post message to pipe dispatcher.
284	Unable to process to pipe dispatcher.
285	Failed to add wait-event to pipe dispatcher.
286	General pipe communication failure.

287	Unable to create pipe (timeout).
288	Call already attached.
289	Async read IO failed.
290	Async write IO failed.
291	Socket attach IO failed.
292	Socket read IO failed.
293	Socket write IO failed.

LIST OF SIP RESPONSES (SIP RFC 3261)

Provisional responses 1xx			
100	Trying	180	Ringing
181	Call Is Being Forwarded	182	Queued
183	Session Progress		
Redirection 3xx			
300	Multiple Choices	301	Moved Permanently
302	Moved Temporarily	305	Use Proxy
380	Alternative Service		
Request Failure 4xx			
400	Bad Request	401	Unauthorized
402	Payment Required	403	Forbidden
404	Not Found	405	Method Not Allowed
406	Not Acceptable	407	Proxy Authentication Required
408	Request Timeout	410	Gone
413	Request Entity Too Large	414	Request-URI Too Long
415	Unsupported Media Type	416	Unsupported URI Scheme
420	Bad Extension	421	Extension Required
423	Interval Too Brief	480	Temporarily Unavailable
481	Call/Transaction Does Not Exist	482	Loop Detected
483	Too Many Hops	484	Address Incomplete
485	Ambiguous	486	Busy Here
487	Request Terminated	488	Not Acceptable Here
491	Request Pending	493	Undecipherable
Server Failure 5xx			
500	Server Internal Error	501	Not Implemented
502	Bad Gateway	503	Service Unavailable
504	Server Time-out	505	Version Not Supported
513	Message Too Large		
Global Failures 6xx			
600	Busy Everywhere	603	Decline
604	Does Not Exist Anywhere	606	Not Acceptable

SIP CLIENT REGISTRATION PROCESS

Please see SIP REC CLIENT REGISTRATION FUNCTIONALITY document for further details.

ACCESS AUDIO DATA PCM PROCESS

Please see ACCESS AUDIO DATA PCM document for further details.