X-Stack (SIP)

- API -

STACK의 구조.

1. 기본 구조.

본 X-Stack은 크게 3개의 Layer로 나누어져 있다고 할 수 있다. 우선 Network으로부터 수신된 Raw Data를 프로그래밍에 사용할 수 있도록 SipMessage 구조체를 Decoding 해주거나 Stack의 처리에 의해 만들어진, 송신될 SipMessage 구조체를 SIP Message Format의 Raw Data로 Encoding 해 주는 기능을 하는 층이 그 첫 번째 이다.

두 번째 layer는 Stack의 현 상태를 기록하고 이를 바탕으로 SipMessage 구조체를 만들어주는 기능을 가진 Layer이다. 여기서는 SIP Protocol의 특성에 맞추어 Stack 정보를 가지는 IStack 구조체는 UA 정보를 가지는 IUa 구조체를 List로 가지고 있으며, IUa 구조체는 다시 호 정보를 가지고 있는 IDialogue 구조체를 List로 가지고 있고, 다시 IDialogue 구조체는 SIP Transaction 정보를 가지고 있는 ITransaction 구조체를 List로 가지고 있다. 전송할 SipMessage를 만들 때에는 이 네가지 구조체의 현재 정보를 이용하게 된다.

세 번째 Layer는 송/수신 된 메시지 또는 Timeout 또는 Programer User로부터 오는 명령들을 입력받아 Stack, UA, 호, Transaction의 상태를 바꾸는 기능을 하는 Layer로써, RFC 3261에 명시된 호처리, Transaction Diagram을 실제로 구현한 층이 된다.

2. 객체의 동작.

여기에서는 H.323과 비교할 때, 다른 점인 객체의 삭제 시점에 대해서 이야기하겠다. 일반적으로 모든 Signaling이 TCP로 이루어져 있으며, 호 Flow의 개념이 하나로 정립되어 있는 H.323 프로토콜의 경우 최종 메시지의 수신 및 발신이 곧 Signaling의 종료를 알렸고, 이와 관련된 객체들은 즉시 삭제 될 수 있었다.

하지만 SIP의 경우 호의 과정이 HTTP와 유사한 Transaction의 집합체로 되어 있으며, H.323의 Endpoint와 유사한 개념을 갖는 UA는, Endpoint와는 조금 다르게 조금더 독자적인 Contact 정보를 가지고 처리된다. 이런 이유로 각각을 List로 갖는 IStack, IUa, IDialogue, ITransaction의 구조체들이 필요하게 되었다.

또한 SIP는 일반적으로 UDP를 사용함으로써 Packet 유실에 대한 보장이 떨어진다. RFC 3261에서는 이를 보충하기 위해, 호를 종료하는 최종 메시지의 수신 및 발신이 이루어진 이후에도 해당 Transaction 및 호의 정보를 유지한 채, 일정 시간동안 기다리도록 권고하고 있다. 이 때문에 Programer User가 예상하는 호 종료 시점에서도 UA, 호, Transaction은 삭제되지 않고 남아 있을 수 있으며, Programer User가 각 객체의 삭제 명령을 하여도 즉시 삭제 할 수가 없다. 본 Stack에서는 이러한 권고 사항을 100% 적용하지는 않았다. 구현 시, 필요 없다고 생각하는 부분들은 생략되었다. 하지만, 각 객체들이 일정시간 남아 있어야 하는 일은 여전히 남아 있으며, 이 때문에, UA와 호가 완전히 종료되었음을 알리는 CallbackUaRemoved, CallbackCallRemoved Callback 함수를 두었다. Application이 어떤 시점에서 호에 대한 정보를 더 이상 관리하지 않을 것인가 하는 문제는 Programer User의 구현상에 판단 문제이지만, 본 X-Stack이 상기와 같이 동작하며, 위에서 언급한 Callback 함수들이 그러한 사실들을 Programer User에게 알려준다는 사실을 알린다.

API 함수.
헤더파일 : SipUI.h

Stack 구동 관련 API

> 함수 원형
HS_STACK_HANDLE SapiMakeStackHandle(char *pLocalIp);
: 앞으로 사용할 Stack의 Handle 하나를 얻어온다.

> Return
만들어진 Stack Handle 반환, 실패시 HS_INVALID_STACK_HANDLE 값 반환.

> Parameter
pLocalIp 자신의 IP 주소로써 IP 형태의 String 값이 들어간다. (예:“192.168.11.20”)

> Remark
: 이 함수를 호출하게 되면 단지 앞으로 사용될 Stack의 Resource를 할당하고 그에 대한 Handle을 반환한 것이고, Stack Thread가 시작 된 것은 아니라는 점에 유의 할 것. 따라서 SapiStartStack 함수를 호출하지 않은 상태에서 지정된 핸들의 Resource를 지우고 싶다면 SapiStopStack 함수를 호출하지 않고 단지 SapiRemoveStackHandle을 호출해 주면 된다.
(SapiStartStack, SapiStopStack, SapiRemoveStackHandle 함수 참조.)

이 함수를 호출하여 Stack Handle을 만든 이후에는 SapiStartStack 함수로 Stack을 구동하기 전에 Programer User가 만들어 놓은 Callback 함수들을 등록해 주어야 한다.
(OESPhone의 AppStartSipStack 함수 참조)

> 함수 원형
HS_RESULT SapiRemoveStackHandle(HS_STACK_HANDLE pHandle);
: 할당된 Stack Handle의 모든 Resource를 해제한다.

> Return
성공시 HS_OK 반환, 실패시 에러코드 반환.

> Parameter
pHandle 할당 받았던 Stack Handle

> Remark
: 단지 할당 되었던 Stack Handle을 해제할 뿐 구동중인 Stack을 멈추는 함수는 아니라는 점을 유의할 것. SapiStartStack 함수를 호출하여 Stack을 구동하기 시작했다면 SapiStopStack 함수를 호출하여 Stack을 멈추어야 한다.
일단 구동된 Stack은 Stop 명령으로 할당된 Resource까지 해제하므로 따로 SapiRemoveStackHande 함수를 호출하지 않고 SapiStopStack 함수만 호출해 주면 된다.

> 함수 원형
```
HS_RESULT SapiStartStack(
  HS_STACK_HANDLE pHandle,
  HS_USHORT pUdpPort,
  HS_USHORT pTcpPort
);
```
: 할당 받았던 Stack을 구동시킨다.

> Return
성공시 HS_OK 반환, 실패시 에러코드 반환.

> Parameter
pHandle 할당 받았던 Stack Handle
pUdpPort SIP UDP Listen Port로 사용할 TSAP Port 번호
pTcpPort SIP TCP Listen Port로 사용할 TSAP Port 번호

> Remark
: 이 함수를 호출하면, 실제 Stack Task가 구동을 시작하게 되며, 이 이후로는 Stack Handle을 API를 호출하는 파라미터 이외의 일을 해서는 안 된다. Stack 구동을 멈추고 싶을 경우 SapiStopStack 함수를 호출하면 되며, Stack에 할당되었던 모든 Resource의 해제도 이 명령을 통해 이루어지므로 따로 Remove 관련 함수를 호출하지 않아도 된다.

> 함수 원형
```
HS_RESULT SapiStopStack(HS_STACK_HANDLE pHandle);
```
: 구동되었던 Stack의 구동을 멈춘다.

> Return
성공시 HS_OK 반환, 실패시 에러코드 반환.

> Parameter
pHandle 구동되었던 Stack Handle

> Remark

: 반드시 구동이 시작된 Stack에 대해서만 호출하면 되며, 이 API명령으로 구동된 Stack의 정지는 물론 기 할당되었던 Resource 까지 해제하여 주므로 따로 Remove 관련 함수를 호출하지 않아도 된다.

UA 관련 API

> 함수 원형

```
IUa *SapiMakeUa(
        char *pProxy,
        char *pOutBoundProxy,
        HS_USHORT pProxyPort,
        HS_UINT pExpires
);
```

: UA 객체에 Resource를 할당하고 할당된 객체를 되돌린다.

> Return

성공시 객체 포인터 반환, 실패시 NULL 반환.

> Parameter

pProxy SIP Proxy 주소, IP 형태나 FQDN형태의 String 입력.
(예:“192.168.11.20” or "www.quenet.co.kr")

pOutBoundProxy OutBoundProxy 주소, IP 형태나 FQDN형태의 String 입력.
(Outbound Proxy에 대해서는 RFC 3261 문서 참고)
(이는 Option사항으로써 사용하지 않을 경우 NULL을 입력하면 된다.)

pProxyPort 접근할 SIP Proxy의 TSAP 주소.
HS_INVALID_TSAP_PORT 입력시, Default 값인 5060이 설정된다.

pExpires Proxy 등록 과정을 시도할 경우, 등록 Timeout value.
(특별히 입력하고 싶은 value가 없다면 0을 입력하면 되고 이때는 Default 값이 1100이 설정된다.)

> Remark

: UA 객체의 Resource만 할당 했을 뿐 UA가 구동하기 시작한 것은 아니라는 점에 유의 해야 한다. 따라서 구동하기 전인 이 상태에서 이 UA를 취소하고 싶을 경우 SapiRemoveUa 함수를 호출하여야 한다.

> 함수 원형

```
HS_RESULT SapiRemoveUa(IUa *pUa);
```

: 할당되었던 UA 객체의 Resource를 해제한다.

> Return

성공시 HS_OK 반환, 실패시 에러코드 반환.

> Parameter
pUa 해제할 UA 객체의 포인터.

> Remark
: 단순히 UA에 할당 된 Resource를 해지하는 동작이다. SapiCommandAddUa 함수를 이용하여 구동시켰던 UA라면 SapiCommandRemoveUa 함수를 호출하여 구동을 멈추어야 한다. 이미 구동된 UA에 정지 명령을 할 경우 이에 관련된 Resource도 내부에서 모두 해지 하므로 이 함수를 따로 호출 할 필요가 없다.

> 함수 원형
```
HS_RESULT SapiAddContactToUa(
    HS_STACK_HANDLE pHandle,
    IUa *pUa,
    char *pDisplayName,
    char *pUserName,
    char *pLocal,
    HS_USHORT pLocalPort,
    HS_UINT pExpires
);
```
: UA 객체에 Contact 정보 하나를 추가한다.

> Return
성공시 HS_OK 반환, 실패시 에러코드 반환.

> Parameter
pHandle Stack Handle.
pUa Contact이 추가 될 UA 객체 포인터.
pDisplayName 추가할 Contact 정보 중 Display Name.
이는 Option 사항으로써 사용하지 않을 경우 NULL을 입력하면 된다.
pUserName 추가할 Contact 정보 중 User Name.
pLocal 추가할 Contact 정보 중 Contact 주소, IP 형태나 FQDN형태의 String 입력.
pLocalPort 추가할 Contact 정보 중 Contact TSAP Port 값.
pExpire 추가할 Contact 정보의 Contact 유지 Timer value.

> Remark
: UA 구동 전에 입력해야 하며 추가된 Contact 정보를 하나씩 제거할 수는 없다.

> 함수 원형

HS_RESULT SapiSetPasswordToUa(IUa *pUa, char *pPassword);

: Authentication 필요시, UA 객체에 Password 값을 설정한다.

> Return

성공시 HS_OK 반환, 실패시 에러코드 반환.

> Parameter

pUa Password가 추가될 UA 객체 포인터.

pPassword 추가할 Password 값.

> Remark

: UA 구동 전에 입력해야 한다.

호 관련 API

> 함수 원형

```
IDialogue *SapiMakeDialogue(
        char *pDisplayName,
        char *pUserName,
        char *pAddress,
        HS_USHORT pPort,
        BOOL pIsTcp
);
```

: 호 객체에 Resource를 할당하고 할당된 호 객체를 되돌린다.

> Return

성공시 객체 포인터 반환, 실패 시 NULL 반환.

> Parameter

pDisplayName Called Party의 DisplayName.
(이는 Option 사항으로, 사용하지 않을 경우 NULL을 입력한다.)
pUserName Called party의 User Name.
pAddress Called party의 주소, IP 형태나 FQDN형태의 String 입력.
pPort Called party의 TSAP port.
(HS_INVALID_TSAP_PORT 입력 시 Default 값인 5060이 적용된다.)

> Remark

: 호 객체의 Resource만 할당 했을 뿐 호 Signaling이 시작된 것은 아니라는 점에 유의해야 한다.
따라서 구동하기 전인 이 상태에서 이 호를 취소하고 싶을 경우 SapiRemoveDialogue 함수를 호출하여야 한다.

> 함수 원형

```
HS_RESULT SapiRemoveDialogue(IDialogue *pDialouge);
```

: 할당되었던 UA 객체의 Resource를 해제한다.

> Return

성공시 HS_OK 반환, 실패시 에러코드 반환.

> Parameter

pDialogue 해지할 호 객체 포인터.

> Remark
: 단순히 호에 할당 된 Resource를 해지하는 동작이다. SapiCommandMakeCall 함수를 이용하여 구동시켰던 호라면 SapiCommandRemoveCall 함수를 호출하여 구동을 멈추어야 한다. 이미 구동된 호에 정지 명령을 할 경우 이에 관련된 Resource도 내부에서 모두 해지 하므로 이 함수를 따로 호출 할 필요가 없다.

> 함수 원형
HS_CALL_HANDLE SapiGetDialogueHandle(IDialogue *pDialogue);
: 호 객체에서 Handle 값을 얻어온다.

> Return
성공시 호 Handle 반환, 실패시 HS_INVALID_HANDLE 반환.

> Parameter
pDialogue Handle을 가져올 호 객체 포인터.

> Remark
: Local에서 발생시키는 호일 경우는 SapiCommandMakeCall의 Return 값으로 호의 Handle을 얻어올 수 있지만, 내가 Called Party일 경우 수신된 호의 관련 객체는 Stack 내부에서 만들어지며 이때, CallbackIncomingCall 함수의 Parameter로 넘어온 호에서 Programer User는 Handle을 얻어 내야만 한다. 이때 이 API를 사용하여 Handle 값을 얻어온다.

> 함수 원형
```
HS_RESULT SapiAddMediaCapabilityToDialogue(
    HS_STACK_HANDLE pHandle,
    IDialogue *pDialogue,
    char *pType,
    HS_UINT pRtpIndex
);
```
: 할당된 호 객체에 Media Capability를 추가한다.

> Return
성공시 HS_OK 반환, 실패시 에러코드 반환.

> Parameter
pHandle Stack Handle.
pDialogue Capability가 추가될 호 객체의 포인터.
pType Media의 Type으로 써 "audio","video","data" 의 식으로 정해주면 된다.
(이는 SDP의 'm' 필드에 사용되며, 하나의 Group으로 묶을 Capability의 구분자로써도 사용된다. Programer User는 필요에 따라 이를 임의로 넣어줄 수 있다.)

pRtpIndex Capability의 구체적인 Codec 정보로써 RTP에서 규정하고 있는 Payload Type 번호가 적용된다.

> Remark

: 본 X-Stack에서는 Media 능력치를 호단위로 넣어주도록 되어 있다. 따라서 매 호가 시작될 시에 이 함수를 이용하여 Capability를 넣어줘야 하며, 수신호가 발생하여 CallbackIncomingCall 함수가 호출 되었을 시에 여기에도 이 API를 이용하여 능력치를 더해줘야 한다.
(OESPhone의 AppCallbackIncomingCall 함수 참고.)
위 함수의 pRtpIndex 파라미터에서 볼 수 있듯이 현재 X-Stack은 RTP에서 규정하는 Payload Type 이외의 Capability의 지정은 아직 지원하지 않고 있다.

UA, 호에 대한 명령 관련 API

(참고, 여기에서 '명령' API를 구분하는 것은 Stack Thread로 Queue Message를 전송하여 동작하는 API를 구분하기 위함이다. '명령' API는 Queue Message의 전달에 대한 성공 여부만 알 수 있다)

> 함수 원형

HS_UA_HANDLE SapiCommandAddUa(HS_STACK_HANDLE pHandle, IUa *pUa);

: UA를 구동시킨다.

> Return

성공시 UA Handle 반환, 실패시 HS_INVALID_HANDLE 반환.

> Parameter

pHandle Stack Handle.

pUa 구동할 UA의 객체 포인터.

> Remark

: 본 X-Stack은 SIP Protocol 구조상 Stack객체가 여러개의 UA객체를 관리하며, UA 객체는 여러개의 호 객체(Dialogue)를 관리하고, 호 객체는 여러개의 Transaction 객체를 관리한다.
UA 객체는 일반적으로 REGISTER를 이용한 등록 시 Contact 헤더 하나에 해당하는 개념으로 프로그래밍 되어 있다.
이 때, 호 객체는 반드시 그를 관리하는 UA 객체가 있어야 한는데, 단대단 호의 경우 UA객체 개념이 애매해진다. 본 Stack에서는 단대단 호를 시도하고 싶을 경우에도 이를 관리하는 UA객체를 만들어주도록 프로그래밍 했다. 따라서 단대단 호를 시도하고 싶을 경우에는 Local의 정보를 넣어 UA 객체를 만들고 본 API 명령을 통해 이를 Stack에 추가시키고, 이 UA를 기준으로 하는 호 객체를 만들고, SapiCommandMakeCall 명령을 통해 호를 구동시키면 된다.
이 때, 이 UA객체는 해당 단대단 호를 위해서만 임시적으로 만들어진 것이므로 호 구동직후, 반드시 SapiCommandRemoveUa API명령을 호출하여, 그 UA의 모든 호가 종료 될 경우 스스로 해제 되도록 해 주어야 한다.
(OESPhone 의 OESPhoneDlg::OnButtonCallHangup 함수의 /*direct call*/ 부분을 추적하여 참고할 것.)
(Stack 내부에서 객체들의 해제 시점에 관련해서는 본 문서의 초입에 설명된 Stack 구조를 참고할 것.)
  본 API 명령을 이용하여 구동된 UA에 대해서는 SapiCommandRemoveUa API 명령을 이용하여 멈추면 되며 이 명령이 시행되면 UA Resource의 해제는 Stack 내부에서 처리하므로 해지 API를 (SapiRemoveUa) 따로 호출 할 필요가 없다.

> 함수 원형

```
HS_RESULT SapiCommandRegistUa(
    HS_STACK_HANDLE pStackHandle,
    HS_UA_HANDLE pUaHandle
);
```

: Handle에 해당하는 UA를 등록 시도한다.

> Return

성공시 HS_OK 반환, 실패시 에러코드 반환.

> Parameter

pStackHandle Stack Handle.
pUaHandle 등록 시도할 UA의 Handle.

> Remark

: SapiCommandAddUa는 단지 UA객체를 Stack에 추가시키는 동작만을 한다.
단대단 호에서는 이 API명령을 호출할 필요가 없다. 일단 이 명령을 호출하면, Stack은 이 UA를 이용하여 한번의 등록시도를 하며, 등록 성공 시에는 CallbackRegisted 함수를 호출하며, Programer User가 지정한, 혹은 Proxy와 협상된 Timeout 시간 간격으로 등록 과정을 시도한다. Reject나 Timeout으로 실패했을 경우 CallbackRegisterFail 함수를 호출하며, Programer User가 등록이 실패하더라도 지속적인 등록 시도를 원할 경우 이 Callback 함수에서 다시 이 API 명령을 이용하여 등록을 시도해야 한다.

이 때, 이 Callback에서 Timer를 두지 않고 바로 시도한다면, Proxy에서 바로 Reject을 하는 경우 프로그램 Load의 위험이 있으므로 Callback에서는 반드시 Timer를 구동하여 수 초 후에 재시도 하도록 할 것을 권장한다.

> 함수 원형

```
HS_RESULT SapiCommandRemoveUa(
    HS_STACK_HANDLE pStackHandle,
    HS_UA_HANDLE pUaHandle
);
```

: Handle에 대당하는 UA를 Stack에서 삭제한다.

> Return

성공시 HS_OK 반환, 실패시 에러코드 반환.

> Parameter

pStackHandle Stack Handle.
pUaHandle 삭제할 UA의 Handle.

> Remark
: 구동이 시작되지 않은 UA에 대해 이 API 명령을 하면 무효 처리된다. 이 함수를 호출했을 때, 해당 UA가 등록 상태라면 등록해지 시도를 한 후 KillMe 상태로 넘어간다.
(KillMe 상태에 대해서는 본 문서의 초입에 설명된 Stack 구조를 참고할 것.)
본 API명령이 실행되면 Stack이 내부적으로 해당 UA에 할당 되었던 Resource를 해지 하므로 따로 Resource 해지 API를 호출할 필요가 없다.
(즉 SapiRemoveUa 함수를 호출해서는 안 된다.)

> 함수 원형

```
HS_CALL_HANDLE SapiCommandMakeCall(
    HS_STACK_HANDLE pStackHandle,
    HS_UA_HANDLE pUaHandle,
    IDialogue *pDialogue
);
```

: 만들어진 해당 호 객체를 이용하여 SIP Call Signaling을 시작한다.

> Return
성공 시 Call Handle 반환, 실패 시 HS_INVALID_HANDLE 반환.

> Parameter
pStackHandle Stack Handle.
pUaHandle 호가 속하게 될 UA의 Handle.
pDialogue 시작할 호 객체 포인터.

> Remark
: 본 X-Stack은 SIP Protocol 구조상 Stack객체가 여러개의 UA객체를 관리하며, UA 객체는 여러개의 호 객체(Dialogue)를 관리하고, 호 객체는 여러개의 Transaction 객체를 관리한다.
UA 객체는 일반적으로 REGISTER를 이용한 등록 시 Contact 헤더 하나에 해당하는 개념으로 프로그래밍 되어 있다.
이 때, 호 객체는 반드시 그를 관리하는 UA 객체가 있어야 한는데, 단대단 호의 경우 UA객체 개념이 애매해진다. 본 Stack에서는 단대단 호를 시도하고 싶을 경우에도 이를 관리하는 UA객체를 만들어주도록 프로그래밍 했다. 따라서 단대단 호를 시도하고 싶을 경우에는 Local의 정보를 넣어 UA 객체를 만들고 본 API 명령을 통해 이를 Stack에 추가시키고, 이 UA를 기준으로 하는 호 객체를 만들고, SapiCommandMakeCall 명령을 통해 호를 구동시키면 된다.
이 때, 이 UA객체는 해당 단대단 호를 위해서만 임시적으로 만들어진 것이므로 호 구동직후, 반드시 SapiCommandRemoveUa API명령을 호출하여, 그 UA의 모든 호가 종료 될 경우 스스로 해제 되도록 해 주어야 한다.
(OESPhone 의 OESPhoneDlg::OnButtonCallHangup 함수의 /*direct call*/ 부분을 추적하여 참고할 것.)
(Stack 내부에서 객체들의 해제 시점에 관련해서는 본 문서의 초입에 설명된 Stack 구조를 참고할 것.)
본 API 명령을 이용하여 구동된 호에 대해서는 SapiCommandRemoveCall API 명령을 이용하여 멈추면 된다.

> 함수 원형

```
HS_RESULT SapiCommandAcceptCall(
    HS_STACK_HANDLE pStackHandle,
    HS_CALL_HANDLE pCallHandle
);
```

: 수신된 호를 Accept 한다.

> Return

성공 시 HS_OK 반환, 실패 시 에러코드 반환.

> Parameter

pStackHandle Stack Handle.

pCallHandle Accept 할호의 Handle.

> Remark

: CallbackCallIncoming 함수가 호출되면 여기에서 SapiGetDialogueHandle API 함수를 이용하여 수신호의 Handle을 얻어오고, Programer User(혹은 실제 사용자)가 Accept하고 싶은 순간에 본 API 명령을 호출하면 된다.
(OESPhone AppCallbackCallIncoming, AppAcceptCall 함수 참조.)

> 함수 원형

```
HS_RESULT SapiCommandRemoveCall(
    HS_STACK_HANDLE pStackHandle,
    HS_CALL_HANDLE pCallHandle,
    SipResponse pReason);
```

: Handle에 해당하는 호를 종료한다.

> Return

성공 시 HS_OK 반환, 실패 시 에러코드 반환.

> Parameter

pStackHandle Stack Handle.

pCallHandle 종료할 호의 Handle.

pResponse 호를 종료하면서 전달할 Response 값.
INVITE 보낸 경우, 호가 시작된 경우에는 각각 CANCEL과 BYE가 전송되 이 값은 무시될 것이지만, INVITE를 받은 상태에서 Response 값을 결정해 준다.
(예를들어 Busy 등의 이유를 정해주기 위해 사용된다. Remark 참조.)

> Remark

: 구동된 호에 대해 명령을 내릴 수 있다. 이 명령이 Stack에 전달되면 Stack은 호 종료 Signal 과정을 시작하고, 종료 Signal이 모두 끝나는 시점에서 호 객체의 resource를 해지한다.
따라서 따로 Resource 해지 관련 API를 호출해 줄 필요가 없다.

SipResponse 값.

```
/*provisional 1xx*/
e_SipResponse_Trying=100,
e_SipResponse_Ringing=180,
e_SipResponse_SessionProgress=183,
/*successful 2xx*/
e_SipResponse_Ok=200,
/*redirection 3xx*/
e_SipResponse_MultipleChoice=300,
e_SipResponse_MovedPermanently=301,
e_SipResponse_MovedTemporary=302,
e_SipResponse_UseProxy=305,
e_SipResponse_AlternativeService=380,
/*request failure 4xx*/
e_SipResponse_BadRequest=400,
e_SipResponse_Unauthorized=401,
e_SipResponse_PaymentRequired=402,
e_SipResponse_Forbidden=403,
e_SipResponse_NotFound=404,
e_SipResponse_MethodNotAllowed=405,
e_SipResponse_NotAcceptable=406,
e_SipResponse_ProxyAuthenticationRequired=407,
e_SipResponse_RequestTimeOut=408,
e_SipResponse_Gone=410,
e_SipResponse_RequestEntityTooLarge=413,
e_SipResponse_Request_URITooLong=414,
e_SipResponse_UnsupportedMediaType=415,
e_SipResponse_UnsupportedURIScheme=416,
e_SipResponse_BadExtension=420,
e_SipResponse_ExtensionRequired=421,
e_SipResponse_IntervalTooBrief=423,
e_SipResponse_TemporarilyUnavailable=480,
e_SipResponse_CallDoesNotExist=481,
e_SipResponse_LoopDetected=482,
e_SipResponse_TooManyHops=483,
e_SipResponse_AddressIncomplete=484,
e_SipResponse_Ambiguous=485,
e_SipResponse_BusyHere=486,
e_SipResponse_RequestTerminated=487,
e_SipResponse_NotAcceptableHere=488,
e_SipResponse_RequestPending=491,
e_SipResponse_Undechiperable=493,
/*server failure 5xx*/
e_SipResponse_ServerInternalError=500,
e_SipResponse_NotImplemented=501,
e_SipResponse_BadGateway=502,
e_SipResponse_ServiceUnavailable=503,
e_SipResponse_ServerTime_Out=504,
e_SipResponse_VersionNotSupported=505,
e_SipResponse_MessageTooLarge=513,
/*global failure 6xx*/
e_SipResponse_BusyEverywhere=600,
e_SipResponse_Decline=603,
e_SipResponse_DoesNotExistAnywhere=604,
e_SipResponse_NotAcceptableEx=606,
```

> 함수 원형

```
HS_RESULT SapiCommandChangeListenPort(
    HS_STACK_HANDLE pStackHandle,
    BOOL pIsUdp,
    HS_USHORT pPort
);
```

: 호 Signaling을 위한 TCP 혹은 UDP의 TSAP Port를 변경한다.

> Return

성공 시 HS_OK 반환, 실패 시 에러코드 반환.

> Parameter

pStackHandle Stack Handle.
pIsUdp TCP/UDP 선택 (TRUE:UDP).
pPort 새로운 TSAP Port 값.

> Remark

: 기존에 진행 중인 호에 대해서는 적용되지 않고, 새로운 호에 대해서부터 적용된다.

부가 Message 관련 API

> 함수 원형

```
HS_RESULT SapiCommandSendINFO(
    HS_STACK_HANDLE pStackHandle,
    HS_CALL_HANDLE pCallHandle,
    char *pType,
    char *pBody
);
```

: 해당 호에 INFO Message Transaction을 시도한다.

> Return

성공 시 HS_OK 반환, 실패 시 에러코드 반환.

> Parameter

pStackHandle Stack Handle.
pCallHandle Call Handle.
pType "Content-Type:" Header에 들어갈 내용
(NULL 입력 시 Default 값은 "application/dtmf-relay")
pBody Message Body에 들어갈 내용.
('₩r₩n'은 입력 시 명기해야 한다.)

> Remark

: 일반적으로 SIP에서 Signaling을 이용한 Outband DTMF 전송시 사용하는 INFO Message Transaction을 시도하는 함수이다. 일반적으로 DTMF 값이 '7'일 경우 아래와 같이 사용한다.

```
SapiCommandSendINFO(sHandle,cHandle,"application/dtmf-relay","signal=7₩r₩n");
```

INFO 메시지 수신 시 Stack은 CallbackOutbandDtmf callback 함수를 호출한다.
(CallbackOutbandDtmf 함수 설명 참조)

> 함수 원형

```
HS_RESULT SapiCommandSendMESSAGE(
    HS_STACK_HANDLE pStackHandle,
    HS_CALL_HANDLE pCallHandle,
    char *pType,
    char *pBody
);
```

: 해당 호에 MESSAGE Message Transaction을 시도한다.

> Return

성공 시 HS_OK 반환, 실패 시 에러코드 반환.

> Parameter

pStackHandle Stack Handle.
pCallHandle Call Handle.
pType "Content-Type:" Header에 들어갈 내용
(NULL 입력 시 Default 값은 "text/plain")
pBody Message Body에 들어갈 내용.
('₩r₩n'은 입력 시 명기해야 한다.)

> Remark

: RFC 3428에서 정의하고 있는 SIP의 Instant Message 전송을 위한 MESSAGE Message Transaction을 시도하는 함수이다. pBody는 ASCII문자중 Visible String 만을 입력으로 받으므로 '한글, 특수문자' 등의 처리는 User Programer가 해 주어야 한다. 일반적으로 'Hello Silver'를 전송할 경우 아래와 같이 사용한다.

SapiCommandSendMESSAGE(sHandle,cHandle,"text/plain","Hello Silver₩r₩n");

MESSAGE Message 수신 시 Stack은 CallbackInstantMessage callback 함수를 호출한다.
(CallbackInstantMessage callback 함수 참조)

Callback 함수.
헤더파일 : IStack.h

메시지 처리 관련 Callback 함수

> 함수 원형
BOOL (*CallbackReceiveRawData)(HS_UCHAR*,int);
: Receive Message가 Decoding 되어 처리되지 직전에 호출됨.

> Return
메시지가 처리되기를 원할 때는 TRUE, 그 이외의 경우 FALSE 반환 함.

> Parameter
HS_UCHAR* 수신된/전송될 Raw Data의 unsigned char pointer.
int 수신된/전송될 Raw Data의 byte size.

> Remark
: 없음.

> 함수 원형
BOOL (*CallbackSendRawData) (HS_UA_HANDLE,HS_CALL_HANDLE,char*,int);
: Send Message가 Encoding 되어 Network의 Raw Data로 날아가기 직전에 호출 됨.

> Return
메시지가 발신되기를 원할 때는 TRUE, 그 이외의 경우 FALSE 반환 함.

> Parameter
HS_UA_HANDLE 전송하는 UA의 Handle.
HS_CALL_HANDLE 전송하는 호의 Handle.
HS_UCHAR* 수신된/전송될 Raw Data의 unsigned char pointer.
int 수신된/전송될 Raw Data의 byte size.

> Remark
: 없음.

> 함수 원형

```
SipResponse (*CallbackReceiveMessageToResponse)(
    HS_UA_HANDLE,
    HS_CALL_HANDLE,
    SipMessage*
);
```

: Response가 필요한 SIP Message 수신 시 발생됨.
(ACK, PRACK을 제외 한 모든 Request Message에 해당한다.)

> Return

Programer User가 응답하고 싶은 SIP Response 값. (SapiCommandRemoveCall 함수 참조.)

> Parameter

HS_UA_HANDLE 수신 메시지에 관련된 UA의 Handle.
HS_CALL_HANDLE 수신 메시지에 관련된 호의 Handle.
SipMessage* Decoding 된 SIP Message 객체 포인터.

> Remark

: Stack에서 해당 메시지를 처리하기 전에 호출하므로 Programer User가 원하는 Field를 변경해 줄 수 있으나, SipMessage 구조체의 특성을 잘 알고 변경해 주어야 한다.

> 함수 원형

```
BOOL (*CallbackReceiveMessage) (HS_UA_HANDLE,HS_CALL_HANDLE,SipMessage*);
```

: Response가 필요 없는 SIP Message 수신 시 발생됨.
(ACK, PRACK와 모든 Response Message에 해당한다.)

> Return

수신 메시지가 처리되기를 원하지 않을 때는 FALSE를, 이외의 경우 TRUE를 반환.

> Parameter

HS_UA_HANDLE 수신 메시지에 관련된 UA의 Handle.
HS_CALL_HANDLE 수신 메시지에 관련된 호의 Handle.
SipMessage* Decoding 된 SIP Message 객체 포인터.

> Remark

: Stack에서 해당 메시지를 처리하기 전에 호출하므로 Programer User가 원하는 Field를 변경해 줄 수 있으나, SipMessage 구조체의 특성을 잘 알고 변경해 주어야 한다.

> 함수 원형

BOOL (*CallbackSendMessage) (HS_UA_HANDLE,HS_CALL_HANDLE,SipMessage*);

: Stack에서 메시지를 보내기 위해 Encoding 하기 직전에 발생 함.

> Return

전송되기를 원할 경우 TRUE, 이외의 경우 FALSE를 반환.

> Parameter

HS_UA_HANDLE 송신 메시지에 관련된 UA의 Handle.

HS_CALL_HANDLE 송신 메시지에 관련된 호의 Handle.

SipMessage* Encoding 될 SIP Message 객체 포인터.

> Remark

: Stack에서 해당 메시지를 Encoding하기 전에 호출하므로 Programer User가 원하는 Field를 변경해 줄 수 있으나, SipMessage 구조체의 특성을 잘 알고 변경해 주어야 한다.

예외 관련 Callback 함수

> 함수 원형

void (*CallbackBnfDecodingError) (HS_UCHAR*,int);

: 수신된 SIP 메시지를 Decoding 할 수 없을 때 발생한다.

> Return

없음.

> Parameter

HS_UCHAR* 수신된 Raw Data의 unsigned char pointer.

int 수신 Raw Data의 byte size.

> Remark

: 없음.

호 Event 관련 Callback 함수

> 함수 원형
  void (*CallbackCallIncoming) (HS_STACK_HANDLE,void*);
  : 원격지로부터 호를 수신 했을 경우 발생.

> Return
  없음.

> Parameter
  HS_STACK_HANDLE Stack Handle.
  void* 호 객체 포인터.

> Remark
  : 호에 Media Capability를 추가해 주기 위해 불가피 하게 IDialogue(호) 객체의 포인터를 void pointer로 넘겨주었다. 여기에서는 Media Capability추가, 호 Handle의 취득 이외에 다른 동작을 해 주어서는 절대로 안 되며, 이 포인터 값을 이후에 다른 곳에 사용해서도 안된다는 것을 반드시 기억해야 한다. (OESPhone AppCallbackCallIncoming 참조)

> 함수 원형
  void (*CallbackCallEnded) (HS_UA_HANDLE,HS_CALL_HANDLE,SipResponse);
  : Local에서 발신된 호가 3XX 이상의 Response로 종료 되었을 경우 호출된다.

> Return
  없음.

> Parameter
  HS_UA_HANDLE 호를 소유하는 UA의 Handle.
  HS_CALL_HANDLE 호의 Handle.
  SipResponse 종료를 발생시킨 3XX 이상의 Response 값.

> Remark
  없음.

> 함수 원형

BOOL (*CallbackCallToRemove) (HS_UA_HANDLE,HS_CALL_HANDLE,RemoveReason);

: 종료 신호를 받은 호를 Remove 할 것인가를 묻기 위한 Callback 함수.

> Return

종료를 원한다면 TRUE 반환, 이외의 경우 FALSE를 반환한다.

> Parameter

HS_UA_HANDLE 호를 소유하고 있는 UA의 Handle.

HS_CALL_HANDLE 호의 Handle.

RemoveReason 호 종료 이유.

> Remark

: SIP는 Protocol 특성상. 호 종료 메시지인 BYE 메시지의 송/수신 후에도, 호를 유지해야 할 경우가 있다. (예: Refer를 이용한 Call Hold) 따라서, 호 종료의 원인이 되는 동작이 발생하여도 Programer User가 호를 종료하지 않고 남겨둘 필요가 있을 것이다. 이를 위해 Remove 여부를 묻는 Callback 함수가 필요하다.

일반적인 경우 TRUE를 반환하여 호를 종료시키면 된다.

> 함수 원형

void (*CallbackCallRemoved) (HS_UA_HANDLE,HS_CALL_HANDLE);

: 호 객체가 Stack에서 완전히 삭제 되었음을 알림.

> Return

없음.

> Parameter

HS_UA_HANDLE 호를 소유하고 있는 UA의 Handle.

HS_CALL_HANDLE 호의 Handle.

> Remark

: SIP Protocol은 HTTP와 비슷하게 Transaction 개념으로 이루어져 있으며, 일반적으로 UDP 기반이기 때문에 RFC 3261에서는 모든 적절한 응답이 끝난 이후에도 Packet 유실을 고려하여 일정시간 동안 Transaction을 유지할 것을 권고하고 있다. 따라서 호 종료 이후에도 이와 관련된 Transaction이 남아 있을 수 있으며, 이를 관리하는 호 객체는 Transaction들이 완전히 종료될 때까지 같이 유지되어야 한다. 따라서 Programer User가 호를 삭제한 순간 이후에도 살아 있을 수 있으며, 이 Callback은 그 모든 Transaction이 끝나 호 객체가 실제적으로 삭제 되었음을 알린다.

부가 Message 관련 Callback 함수

> 함수 원형

HS_RESULT (*CallbackOutbandDtmf) (HS_UA_HANDLE,HS_CALL_HANDLE,char*,char*);

: 상대방으로부터 INFO 메시지를 받았을 경우 발생한다.

> Return

없음.

> Parameter

HS_UA_HANDLE	메시지에 관련 된 UA Handle.
HS_CALL_HANDLE	메시지에 관련된 Call handle.
char*	'Content-Type' 헤더 정보.
char*	INFO의 Message Body.

> Remark

: 일반적으로 SIP의 Outband DTMF 전송에 쓰이는 INFO 메시지를 받았을 경우 호출된다.
일반적으로 전송 된 DTMF가 '7'의 값일 경우 Message Body는 'signal=7₩r₩n'이 된다.
(SapiCommandSendINFO 함수 참조.)

HS_RESULT (*CallbackInstantMessage) (HS_UA_HANDLE,HS_CALL_HANDLE,char*,char*);

: 상대방으로부터 INFO 메시지를 받았을 경우 발생한다.

> Return

없음.

> Parameter

HS_UA_HANDLE	메시지에 관련 된 UA Handle.
HS_CALL_HANDLE	메시지에 관련된 Call handle.
char*	'Content-Type' 헤더 정보.
char*	MESSAGE의 Message Body.

> Remark

: RFC 3428에서 정의하고 있는 SIP의 Instant Message 전송을 위한 MESSAGE Message 를 수신 했을 때 호출되는 함수이다.
(SapiCommandSendMESSAGE 함수 참조.)

Media 관련 Callback 함수

> 함수 원형

HS_RESULT (*CallbackCheckSdp) (void*,SdpMessage*); /*IDialogue object pointer*/

: 상대방으로부터 SDP 메시지를 받았을 경우 발생한다.

> Return

Stack이 따로 Check 하기를 원하지 않을 경우(Programer User가 적절한 처리를 했을 경우) HS_OK 반환, 이외의 경우 HS_ERR를 반환하여 Stack이 Check하도록 한다.

> Parameter

void* 호 객체 포인터.

SdpMessage* 수신된 SDP Message 구조체 포인터.

> Remark

: HS_ERR 반환 시, Stack에서는 Programer User가 입력해 놓은 Media Capability와 수신 SDP를 비교하여 적절한 Media를 찾을 것이다. 일반적인 처리와 다르게 처리하고 싶을 경우 Programer User는 SDP 메시지에서 적절히 Capability 발견 절차를 밟아야 하며, 발견된 Capability를 호 객체에 넣어주어야 한다. 따라서 객체 특성을 정확하게 알기 전에는 HS_ERR를 반환하여 일반적인 처리 과정을 거치도록 해야 한다.

> 함수 원형

void (*CallbackOpenMedia) (void*); /*MediaInfo object pointer*/

void (*CallbackCloseMedia) (void*); /*MediaInfo object pointer*/

: SDP 협상에 의해 Media가 시작/종료 되어야 하는 시점에서 발생한다.

> Return

없음.

> Parameter

void* MediaInfo 구조체의 포인터.

> Remark

: 이 Callback 함수에서는 void*를 MediaInfo*로 Type cast 하여 처리한다.
MediaInfo 구조체는 시작/종료될 Media의 Type 정보가 들어가 있으며, 이는 Programer User가 SapiAddMediaCapabilityToDialogue API에서 세 번째 파라미터로 넣어준 값이다.
또한 자세한 Codec 종류와 이를 주고 받을 TSAP 주소가 정의되어 있다.
(현재는 RTP관련 Media만을 처리하기 때문에 구조체는 이에 맞게 구성되어 있다.)

다음은 MediaInfo 구조체의 구조이다.

```
typedef struct
{
    HS_UINT mNumber;        // RTP Payload Type Number
    char    *mType;         // Media Type (audio,video,data)

    char            *mLocalRtpAddr;         // Local Listen Address
    HS_USHORT       mLocalRtpPort;          // Local Listen TSAP Port
    char            *mRemoteRtpAddr;        // Remote Listen Address
    HS_USHORT       mRemoteRtpPort;         // Remote Listen Address
} MediaInfo;
```

UA 관련 Callback 함수

> 함수 원형
  void (*CallbackRegisted) (HS_UA_HANDLE);
  : 해당 UA가 등록에 성공 했을 경우 발생.

> Return
  없음.

> Parameter
  HS_UA_HANDLE 등록에 성공한 UA Handle.

> Remark
  : 최초 등록 시에만 발생하며, Timeout Expires 에 의한 등록 시도와 성공시에는 발생하지 않는다.

> 함수 원형
  void (*CallbackRegisterFail) (HS_UA_HANDLE);
  : 해당 UA가 등록에 실패 했을 경우 발생.

> Return
  없음.

> Parameter
  HS_UA_HANDLE 등록에 실패한 UA Handle.

> Remark
  : SapiCommandRegistUa API 명령에 의해 시도되는 Register는 등록시도를 한번만 하게 되며, Reject/Timeout 등의 이유로 등록이 실패했을 경우 이 Callback 함수를 호출한 후 아무것도 하지 않는다. 실패 하더라도 지속적인 등록 시도를 원할 경우 Programer User는 이 Callback 함수를 기준으로 다시 SapiCommandRegistUa API를 호출하여야 한다.
    이 때, 이 Callback에서 Timer를 두지 않고 바로 시도한다면, Proxy에서 바로 Reject을 하는 경우 프로그램 Load의 위험이 있으므로 Callback에서는 반드시 Timer를 구동하여 수 초 후에 재시도 하도록 할 것을 권장한다. (OESPhone 의 AppCallbackRegisterFail 함수 참조.)

> 함수 원형

void (*CallbackUaRemoved) (HS_UA_HANDLE);

: 해당 UA가 Stack에서 완전히 삭제 되었음을 알림.

> Return

없음.

> Parameter

HS_UA_HANDLE 삭제된 UA Handle.

> Remark

: CallbackCallRemoved 함수에서 설명했듯이 Stack은 UA를 UA는 호를 호는 Transaction을 가지고 관리한다. 따라서 Programer User가 SapiCommandRemoveUa API를 이용하여 UA 삭제 명령을 내렸다고 해도 해당 UA에 진행중인 호나 Transaction이 있을 경우 이를 바로 삭제하지 않는다. 따라서 해당 UA에 모든 호나 Transaction이 끝나는 순간 UA가 삭제되며 이를 알리기 위해 본 Callback 함수를 사용한다.