



# CHAPTER 6

## Cisco Device-Specific Extensions

This chapter describes the Cisco device-specific TAPI extensions. `CCiscoLineDevSpecific` and the `CCiscoPhoneDevSpecific` class represent the parent class. This chapter describes how to invoke the Cisco device-specific TAPI extensions with the `lineDevSpecific` function. It also describes a set of classes that you can use when you call `phoneDevSpecific`. It contains the following sections:

- [Cisco Line Device Specific Extensions, page 6-1](#)
- [Cisco Line Device Feature Extensions, page 6-61](#)
- [Cisco Phone Device-Specific Extensions, page 6-64](#)
- [Messages, page 6-71](#)

### Cisco Line Device Specific Extensions

[Table 6-1](#) lists the subclasses of Cisco Line Device-Specific Extensions. This section contains all of the extensions in the table and descriptions of the following data structures:

- [LINEDEVCAPS, page 6-4](#)
- [LINECALLINFO, page 6-8](#)
- [LINEDEVSTATUS, page 6-19](#)

**Table 6-1** Cisco Line Device-Specific Extensions

Cisco Functions	Synopsis
<a href="#">CCiscoLineDevSpecific</a>	The <code>CCiscoLineDevSpecific</code> class specifies the parent class to the following classes.
<a href="#">Message Waiting</a>	The <code>CCiscoLineDevSpecificMsgWaiting</code> class turns the message waiting lamp on or off for the line that the <code>hLine</code> parameter specifies.
<a href="#">Message Waiting Dirn</a>	The <code>CCiscoLineDevSpecificMsgWaiting</code> class turns the message waiting lamp on or off for the line that a parameter specifies and remains independent of the <code>hLine</code> parameter.
<a href="#">Message Summary</a>	The <code>CCiscoLineDevSpecificSetMsgSummary</code> class turns the message waiting lamp on or off, as well as provides voice and fax message counts for the line specified by the <code>hLine</code> parameter.

Table 6-1 Cisco Line Device-Specific Extensions (continued)

Cisco Functions	Synopsis
Message Summary Dirn	The CCiscoLineDevSpecificSetMsgSummaryDirn class turns the message waiting lamp on or off and provides voice and fax message counts for the line specified by a parameter and is independent of the hLine parameter.
Audio Stream Control	The CCiscoLineDevSpecificUserControlRTPStream class controls the audio stream for a line.
Set Status Messages	The CCiscoLineDevSpecificSetStatusMsgs class controls the reporting of certain line device specific messages for a line.
Swap-Hold/SetupTransfer	Cisco Unified TSP 4.0 and later do not support this function. The CCiscoLineDevSpecificSwapHoldSetupTransfer class performs a setupTransfer between a call that is in CONNECTED state and a call that is in ONHOLD state. This function will change the state of the connected call to ONHOLDPENDTRANSFER state and the ONHOLD call to CONNECTED state. This action will then allow a completeTransfer to be performed on the two calls.
Redirect Reset Original Called ID	The CCiscoLineDevSpecificRedirectResetOrigCalled class gets used to redirect a call to another party while resetting the original called ID of the call to the destination of the redirect.
Port Registration per Call	The CciscoLineDevSpecificPortRegistrationPerCall class gets used to register a CTI port or route point for the Dynamic Port Registration feature, which allows applications to specify the IP address and UDP port number on a call-by-call basis.
Setting RTP Parameters for Call	The CciscoLineDevSpecificSetRTPParamsForCall class sets the IP address and UDP port number for the specified call.
Redirect Set Original Called ID	The CCiscoLineDevSpecificRedirectSetOrigCalled class to redirect a call to another party while setting the original called ID of the call to any other party.
Join	The CciscoLineDevSpecificJoin class joins two or more calls into one conference call.
Set User SRTP Algorithm IDs	The CciscoLineDevSpecificUserSetSRTPAlgorithmID class allows the application to set SRTP algorithm IDs. You should use this class after lineopen and before CCiscoLineDevSpecificSetRTPParamsForCall or CCiscoLineDevSpecificUserControlRTPStream
Explicit Acquire	The CciscoLineDevSpecificAcquire class explicitly acquires any CTI Controllable device in the Cisco Unified Communications Manager system, which needs to be opened in Super Provider mode.
Explicit De-Acquire	The CciscoLineDevSpecificDeacquire class explicitly de-acquires any CTI controllable device in the Cisco Unified Communications Manager system.
Redirect FAC CMC	The CCiscoLineDevSpecificRedirectFACCMC class redirects a call to another party while including a FAC, CMC, or both.
Blind Transfer FAC CMC	The CCiscoLineDevSpecificBlindTransferFACCMC class blind transfers a call to another party while including a FAC, CMC, or both.

**Table 6-1 Cisco Line Device-Specific Extensions (continued)**

<b>Cisco Functions</b>	<b>Synopsis</b>
<a href="#">CTI Port Third Party Monitor</a>	The CCiscoLineDevSpecificCTIPortThirdPartyMonitor class opens a CTI port in third-party mode.
<a href="#">Send Line Open</a>	The CciscoLineDevSpecificSendLineOpen class triggers actual line open from TSP side. Use this for delayed open mechanism.
<a href="#">Set Intercom SpeedDial</a>	The CciscoLineSetIntercomSpeeddial class allows the application to set or reset SpeedDial/Label on an intercom line.
<a href="#">Intercom Talk Back</a>	The CciscoLineIntercomTalkback class allows the application to initiate talk back on an incoming Intercom call on an Intercom line.
<a href="#">Redirect with Feature Priority</a>	The CciscoLineRedirectWithFeaturePriority class enables the application to redirect calls with specified priority.
<a href="#">Start Call Monitoring</a>	The CCiscoLineDevSpecificStartCallMonitoring class allows applications to send a start monitoring request for the active call on a line.
<a href="#">Start Call Recording</a>	The CCiscoLineDevSpecificStartCallRecording allows the application to send a recording request for the active call on that line.
<a href="#">StopCall Recording</a>	The CCiscoLineDevSpecificStopCallRecording allows the application to stop recording a call on that line.
<a href="#">Set IPv6 Address and Mode</a>	The CciscoLineDevSpecificSetIPv6AddressAndMode enables the application to set the IPv6 address and addressing mode during registration.
<a href="#">Set RTP Parameters for IPv6 Calls</a>	The CciscoLineDevSpecificSetRTTPParamsForCallIPv6 class sets the RTP parameters for calls for which you must specify IPv6 address.
<a href="#">Direct Transfer</a>	The CciscoLineDevSpecificDirectTransfer class transfers calls across lines or on the same line.
<a href="#">RegisterCallPickUpGroup ForNotification</a>	The CciscoLineDevSpecificRegisterCallPickupGroupForNotification class is used to register the call Pickup Group for notification on calls for Pickup.
<a href="#">UnRegisterCallPickUpGro upForNotification</a>	The CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification class is used to unregister the call Pickup Group for notification on calls for Pickup.
<a href="#">CallPickUpRequest</a>	This feature allows to invoke the pickup, group-pickup, other-pickup, and directed pickup feature from the application. Apart from providing API to invoke feature, application will have capability to register Call pickup group for alert notification, whenever a call is available for pickup.  The CciscoLineDevSpecificCallPickupRequest class is used to Pickup the call from the PickGroup.
<a href="#">Start Send Media To BIB</a>	The CCiscoLineDevSpecificStartSendMediaToBIBRequest class allows the application to initiate agent greeting to the customer call.
<a href="#">Stop Send Media To BIB</a>	The CCiscoLineDevSpecificStopSendMediaToBIBRequest class allows the application to stop agent greeting that is playing on the agent-to-customer call.
<a href="#">Agent Zip Tone</a>	The CciscoLineDevSpecificEnableFeatureSupport class allows the application to initiate Zip tone on the Agent Call.

**Table 6-1 Cisco Line Device-Specific Extensions (continued)**

Cisco Functions	Synopsis
Enable Feature	The CciscoLineDevSpecificEnableFeatureSupport class allows the application to enhance or update feature support.
Transfer with media	The CciscoSetupTransferWithoutMedia class allows the application to transfer a call that does not have media setup.

## LINEDEVCAPS

Cisco TSP implements several line device-specific extensions and uses the DevSpecific (dwDevSpecificSize and dwDevSpecificOffset) variably sized area of the LINEDEVCAPS data structure for those extensions. The the Cisco\_LineDevCaps\_Ext structure in the CiscoLineDevSpecificMsg.h header file defines the DevSpecific area layout. Cisco TSP organizes the data in that structure based on the extension version in which the data was introduced:

```
// LINEDEVCAPS Dev Specific extention //
typedef struct Cisco_LineDevCaps_Ext
{
    Cisco_LineDevCaps_Ext00030000  ext30;
    Cisco_LineDevCaps_Ext00060000  ext60;
    Cisco_LineDevCaps_Ext00070000  ext70;
    Cisco_LineDevCaps_Ext00080000  ext80;
    Cisco_LineDevCaps_Ext00090000  ext90;
    Cisco_LineDevCaps_Ext00090001  ext91;
    Cisco_LineDevCaps_Ext000A0000  extA0;
} CISCO_LINEDEVCAPS_EXT;
```

For a specific line device, the extension area will include a portion of this structure starting from the beginning and up to the extension version that an application negotiated.

The individual extension version substructure definitions follow:

```
// LINEDEVCAPS 00030000 extention //
typedef struct Cisco_LineDevCaps_Ext00030000
{
    DWORD dwLineTypeFlags;
} CISCO_LINEDEVCAPS_EXT00030000;
// LINEDEVCAPS 00060000 extention //
typedef struct Cisco_LineDevCaps_Ext00060000
{
    DWORD dwLocale;
} CISCO_LINEDEVCAPS_EXT00060000;
// LINEDEVCAPS 00070000 extention //
typedef struct Cisco_LineDevCaps_Ext00070000
{
    DWORD dwPartitionOffset;
    DWORD dwPartitionSize;
} CISCO_LINEDEVCAPS_EXT00070000;
// LINEDEVCAPS 00080000 extention //
typedef struct Cisco_LineDevCaps_Ext00080000
{
    DWORD dwLineDevCaps_DevSpecificFlags; //
LINEFEATURE_DEVSPECIFIC
    DWORD dwLineDevCaps_DevSpecificFeatureFlags; //
LINEFEATURE_DEVSPECIFICFEAT
    RECORD_TYPE_INFO recordTypeInfo;
    INTERCOM_SPEEDDIAL_INFO intercomSpeedDialInfo;
} CISCO_LINEDEVCAPS_EXT00080000;
```

```

// LINEDEVCAPS 00090000 extention //
// -----
typedef struct Cisco_LineDevCaps_Ext00090000
{
    IpAddressingMode      dwLineDevCapsIPAddressingMode;      // LINEFEATURE_DEVSPECIFIC
} CISCO_LINEDEVCAPS_EXT00090000;

//=====
//          Cisco Extention 00090001
//=====
// LINEDEVCAPS 00090001 extention //
// -----
typedef struct Cisco_LineDevCaps_Ext00090001
{
    DWORD    MaxCalls ;
    DWORD    BusyTrigger ;
    DWORD    LineInstanceNumber ;
    DWORD    LineLabelASCIIOffset ;
    DWORD    LineLabelASCIISize ;
    DWORD    LineLabelUnicodeOffset ;
    DWORD    LineLabelUnicodeSize ;
    DWORD    VoiceMailPilotDNOffset ;
    DWORD    VoiceMailPilotDNSize ;
    DWORD    RegisteredIPAddressMode;// IpAddressingMode
            DWORD    RegisteredIPv4Address ;
            DWORD    RegisteredIPv6AddressOffset;
    DWORD    RegisteredIPv6AddressSize;
    DWORD    ApplicationFeatureFlagBitMap;// CiscoFeatureInformation
    DWORD    DeviceFeatureFlagBitMap; // CiscoFeatureInformation
} CISCO_LINEDEVCAPS_EXT00090001;

typedef struct Cisco_LineDevCaps_Ext000A0000
{
    DWORD    dwPickUpGroupDNOffset;
    DWORD    dwPickUpGroupDNSize;
    DWORD    dwPickUpGroupPartitionOffset;
    DWORD    dwPickUpGroupPartitionSize;
} CISCO_LINEDEVCAPS_EXT000A0000;

```

See the `CiscoLineDevSpecificMsg.h` header file for additional information on the `DevSpecific` structure layout and data.

## Detail

### A

```

typedef struct LineDevCaps_DevSpecificData
{
    DWORD    m_DevSpecificFlags;
}LINEDEVCAPS_DEV_SPECIFIC_DATA;

```



#### Note

Be aware that this extension is only available if extension version 3.0 (0x00030000) or higher is negotiated.

### B

```

typedef struct LocaleInfo
{
    DWORD    Locale; //This will have the locale info of the device
    DWORD    PartitionOffset;
    DWORD    PartitionSize; //This will have the partition info of the line.
}

```

```
} LOCALE_INFO;
```

**Note**

Be aware that the Locale info is only available along with LINEDEVCAPS\_DEV\_SPECIFIC\_DATA if extension version 6.0 (0x00060000) or higher is negotiated.

**C**

```
typedef struct PartitionInfo
{
    DWORD PartitionOffset;
    DWORD PartitionSize; //This will have the partition info of the line.
} PARTITION_INFO;
```

**Note**

Be aware that both the Locale and Partition Info is available along with LINEDEVCAPS\_DEV\_SPECIFIC\_DATA if extension version 6.1 (0x00060001) or higher is negotiated.

**D**

```
typedef struct Intercom_Speeddial_Info
{
    DWORD IsIntercomSpeeddialOverridden; //indicating whether the Intercom Speeddial value
has been overridden by applicaiton
    //Intercom setting in CCM database
    DWORD DBIntercomInfoStructureOffset; // offset from base of LINECALLINFO
    DWORD DBIntercomInfoStructureSize; // includes variable length data total size
    DWORD DBIntercomInfoStructureElementCount;
    DWORD DBIntercomInfoStructureElementFixedSize;
    //Current Intercom setting, if different than DB setting, then it is set by CTI
application
    DWORD CurrIntercomInfoStructureOffset; // offset from base of LINECALLINFO
    DWORD CurrIntercomInfoStructureSize; // includes variable length data total size
    DWORD CurrIntercomInfoStructureElementCount;
    DWORD CurrIntercomInfoStructureElementFixedSize;
} INTERCOM_SPEEDDIAL_INFO;

typedef struct IntercomSpeeddialInfoStruct
{
    DWORD IntercomSpeeddialDNOffset;
    DWORD IntercomSpeeddialDNSize;
    DWORD IntercomSpeeddialAsciiNameOffset;
    DWORD IntercomSpeeddialAsciiNameSize;
    DWORD IntercomSpeeddialUnicodeNameOffset;
    DWORD IntercomSpeeddialUnicodeNameSize;
} IntercomSpeeddialInfoStruct;
```

**E**

```
typedef struct RecordingTypeInfo
{
    DWORD RecordingType;
} RECORDING_TYPE_INFO;
```

**Note**

Recording Type information is available along with other devSpecific data if extension version 8.0 (0x00080000) or higher is negotiated.

**F**

```
enum IpAddressingMode{
    IPAddress_IPv4_only = 0,
    IPAddress_IPv6_only = 1,
    IPAddress_IPv4_IPv6 = 2,
    IPAddress_Unknown = 3,
    IPAddress_unknown_ANATRed
};
```

**G**

```
enum CiscoFeatureInformation
{
    DWORD NewCallRollOverEnabled = 0x00000001;
    DWORD ConsultCallRollOverEnabled = 0x00000002;
    DWORD JoinOnSameLineEnabled = 0x00000004;
    DWORD JoinAcrossLineEnabled = 0x00000008;
    DWORD DirectTransferSameLineEnabled = 0x00000010;
    DWORD DirectTransferAcrossLineEnabled = 0x00000020;
} CISCO_FEATURE_INFORMATION;
```

## Parameters

### DWORD m\_DevSpecificFlags

A bit array that identifies device-specific properties for the line. The bits definition follows:

LINEDEVCAPSDEVSPECIFIC\_PARKDN (0x00000001)—Indicates whether this line is a Call Park DN.



**Note** Be aware that this extension is only available if extension version 3.0 (0x00030000) or higher is negotiated.

### DWORD Locale

This entity identifies the locale information for the device. The typical values could be:

```
enum
{
    ENGLISH_UNITED_STATES= 1,
    FRENCH_FRANCE= 2,
    GERMAN_GERMANY= 3,
    RUSSIAN_RUSSIAN_FEDERATION= 5,
    SPANISH_SPAIN= 6,
    ITALIAN_ITALY= 7,
    DUTCH_NETHERLANDS= 8,
    NORWEGIAN_NORWAY= 9,
    PORGUGUESE_PORTUGAL= 10,
    SWEDISH_SWEDEN= 11,
    DANISH_DENMARK= 12,
    JAPANESE_JAPAN= 13,
    HUNGARAIN_HUNGARY= 14,
    POLISH_POLAND= 15,
    GREEK_GREECE= 16,
    CHINESE_TAIWAN = 19,
    CHINESE_CHINA= 20,
    KOREAN_KOREA_REPUBLIC= 21,
```

```

FINNISH_FINLAND= 22,
PORTUGUESE_BRAZIL= 23,
CHINESE_HONG_KONG= 24,
SLOVAK_SLOVAKIA= 25,
CZECH_CZECH_REPUBLIC= 26,
BULGARIAN_BULGARIA= 27,
CROATIAN_CROATIA= 28,
SLOVENIAN_SLOVENIA= 29,
ROMANIAN_ROMANIA= 30,
CATALAN_SPAIN= 32,
ENGLISH_UNITED_KINGDOM= 33,
ARABIC_UNITED_ARAB_EMIRATES= 35,
ARABIC_OMAN= 36,
ARABIC_SAUDI_ARABIA= 37,
ARABIC_KUWAIT= 38,
HEBREW_ISRAEL= 39,
SERBIAN_REPUBLIC_OF_SERBIA= 40,
SERBIAN_REPUBLIC_OF_MONTENEGRO= 41,
THAI_THAILAND= 42,
ARABIC_ALGERIA= 47,
ARABIC_BAHRAIN= 48,
ARABIC_EGYPT= 49,
ARABIC_IRAQ= 50,
ARABIC_JORDAN= 51,
ARABIC_LEBANON= 52,
ARABIC_MOROCCO= 53,
ARABIC_QATAR= 54,
ARABIC_TUNISIA= 55,
}

```

## LINECALLINFO

Cisco TSP implements several line device-specific extensions and uses the DevSpecific (dwDevSpecificSize and dwDevSpecificOffset) variably sized area of the LINECALLINFO data structure for those extensions. The Cisco\_LineCallInfo\_Ext structure in the CiscoLineDevSpecificMsg.h header file defines DevSpecific area layout. Cisco TSP organizes the data in that structure based on the extension version in which the data was introduced:

```

// LINECALLINFO Dev Specific extention //
typedef struct Cisco_LineCallInfo_Ext
{
    Cisco_LineCallInfo_Ext00060000 ext60;
    Cisco_LineCallInfo_Ext00070000 ext70;
    Cisco_LineCallInfo_Ext00080000 ext80;
    Cisco_LineCallInfo_Ext00080001 ext81;
    Cisco_LineCallInfo_Ext00090000ext90;
    Cisco_LineCallInfo_Ext000A0000 extA0;

} CISCO_LINECALLINFO_EXT;

```

For a specific line device, the extension area includes a portion of this structure starting from the beginning and up to the extension version that an application negotiated.

The individual extension version substructure definitions follow:

```

// LINECALLINFO 00060000 extention //
typedef struct Cisco_LineCallInfo_Ext00060000
{
    TSP_UNICODE_PARTY_NAMES unicodePartyNames;
} CISCO_LINECALLINFO_EXT00060000;
// LINECALLINFO 00070000 extention //

```



```

typedef struct Cisco_LineCallInfo_Ext00070000
{
    DWORD SRTPKeyInfoStructureOffset;    // offset from base of LINECALLINFO
    DWORD SRTPKeyInfoStructureSize;     // includes variable length data total size
    DWORD SRTPKeyInfoStructureElementCount;
    DWORD SRTPKeyInfoStructureElementFixedSize;
    DWORD DSCPInformationOffset;        // offset from base of LINECALLINFO
    DWORD DSCPInformationSize;         // fixed size of the DSCPInformation structure
    DWORD DSCPInformationElementCount;
    DWORD DSCPInformationElementFixedSize;
    DWORD CallPartitionInfoOffset;     // offset from base of LINECALLINFO
    DWORD CallPartitionInfoSize;      // fixed size of the CallPartitionInformation
    structure
    DWORD CallPartitionInfoElementCount;
    DWORD CallPartitionInfoElementFixedSize;
    DWORD ExtendedCallInfoOffset;     // ==> ExtendedCallInfo { }
    DWORD ExtendedCallInfoSize;       //
    DWORD ExtendedCallInfoElementCount; //
    DWORD ExtendedCallInfoElementSize; //
} CISCO_LINECALLINFO_EXT00070000;
// LINECALLINFO 00080000  extension //
// -----
typedef struct Cisco_LineCallInfo_Ext00080000
{
    DWORD CallSecurityStatusOffset;
    DWORD CallSecurityStatusSize;
    DWORD CallSecurityStatusElementCount;
    DWORD CallSecurityStatusElementFixedSize;
    DWORD CCMCallIDInfoOffset;
    DWORD CCMCallIDInfoSize;
    DWORD CCMCallIDInfoElementCount;
    DWORD CCMCallIDInfoElementFixedSize;
    DWORD CallAttributeInfoOffset;
    DWORD CallAttributeInfoSize;
    DWORD CallAttributeInfoElementCount;
    DWORD CallAttributeInfoElementFixedSize;
    DWORD TSPIntercomSideInfo;
    DWORD CallingPartyIpAddr;
} CISCO_LINECALLINFO_EXT00080000;
// LINECALLINFO 00080001  extension //
// -----
typedef struct Cisco_LineCallInfo_Ext00080001
{
    DWORD CPNInfoOffset;              //array of structure of CPNInfo structure
    DWORD CPNInfoSize;
    DWORD CPNInfoElementCount;
    DWORD CPNInfoElementFixedSize;
} CISCO_LINECALLINFO_EXT00080001;
// LINECALLINFO 00090000  extension //
// -----
typedef struct Cisco_LineCallInfo_Ext00090000
{
    DWORD IPv6InfoOffset;
    DWORD IPv6InfoSize;
    DWORD IPv6InfoElementCount;
    DWORD IPv6InfoElementFixedSize;
    DWORD FarEndIPAddressingMode;
}CISCO_LINECALLINFO_EXT00090000;
// LINECALLINFO 000A0000  extension //
// -----
typedef struct Cisco_LineCallInfo_Ext000A0000
{
    DWORD CallAttributeBitMask;

```

```

    DWORD UniqueCallRefIDInfoOffset;
    DWORD UniqueCallRefIDInfoSize;
    DWORD UniqueCallRefIDInfoElementCount;
    DWORD UniqueCallRefIDInfoElementFixedSize;
    //HuntList
    DWORD HuntPilotInfoOffset; //point to HuntPilotInfo
    DWORD HuntPilotInfoSize;
    DWORD HuntPilotInfoCount;
    DWORD HuntPilotInfoElementFixedSize;
    DWORD GlobalCallID;
    DWORD CallManagerID;
} CISCO_LINECALLINFO_EXT000A0000;

```

See the CiscoLineDevSpecificMsg.h header file for additional information on the DevSpecific structure layout and data.

## Details

The TSP\_Unicode\_Party\_names structure and SRTP information structure describe the device-specific extensions that the Cisco Unified TSP made to the LINECALLINFO structure.

DSCPValueForAudioCalls will contain the DSCP value that CTI sent in the StartTransmissionEvent.

ExtendedCallInfo structure has extra call information. For Cisco Unified Communications Manager Release 7.0(1), the ExtendedCallReason field belongs to the ExtendedCallInfo structure.

CallAttributeInfo contains the information about attributeType (Monitoring, Monitored, Recorder, securityStatus) and PartyInfo (Dn, Partition, DeviceName)

CCMCallID contains CCM Call identifier value.

CallingPartyIPAddress contains the IP address of the calling party if the calling party device supports it.

CallSecurityStatus structure contains the overall security status of the call for two-party call as well as conference call.

```

DWORD TapiCallerPartyUnicodeNameOffset;
    DWORD TapiCallerPartyUnicodeNameSize;
    DWORD TapiCallerPartyLocale;

    DWORD TapiCalledPartyUnicodeNameOffset;
    DWORD TapiCalledPartyUnicodeNameSize;
    DWORD TapiCalledPartyLocale;

    DWORD TapiConnectedPartyUnicodeNameOffset;
    DWORD TapiConnectedPartyUnicodeNameSize;
    DWORD TapiConnectedPartyLocale;

    DWORD TapiRedirectionPartyUnicodeNameOffset;
    DWORD TapiRedirectionPartyUnicodeNameSize;
    DWORD TapiRedirectionPartyLocale;

    DWORD TapiRedirectingPartyUnicodeNameOffset;
    DWORD TapiRedirectingPartyUnicodeNameSize;
    DWORD TapiRedirectingPartyLocale;

    DWORD SRTPKeyInfoStructureOffset; // offset from base of LINECALLINFO
    DWORD SRTPKeyInfoStructureSize; // includes variable length data total size
    DWORD SRTPKeyInfoStructureElementCount;
    DWORD SRTPKeyInfoStructureElementFixedSize;
    DWORD DSCPValueInformationOffset;
    DWORD DSCPValueInformationSize;
    DWORD DSCPValueInformationElementCount;
    DWORD DSCPValueInformationElementFixedSize;

```

```

DWORD PartitionInformationOffset; // offset from base of LINECALLINFO
DWORD PartitionInformationSize; // includes variable length data total size
DWORD PartitionInformationElementCount;
DWORD PartitionInformationElementFixedSize;
DWORD ExtendedCallInfoOffset;
DWORD ExtendedCallInfoSize;
DWORD ExtendedCallInfoElementCount;
DWORD ExtendedCallInfoElementSize;
DWORD CallAttributeInfoOffset;
DWORD CallAttributeInfoSize;
DWORD CallAttributeInfoElementCount;
DWORD CallAttributeInfoElementSize;
DWORD CallingPartyIPAddress;
DWORD CCMCallIDInfoOffset;
DWORD CCMCallIDInfoSize;
DWORD CCMCallIDInfoElementCount;
DWORD CCMCallIDInfoElementFixedSize;
DWORD CallSecurityStatusOffset;
DWORD CallSecurityStatusSize;
DWORD CallSecurityStatusElementCount;
DWORD CallSecurityStatusElementFixedSize;
DWORD IsChaperoneCall;
DWORD UniqueCallRefIDInfoOffset;
DWORD UniqueCallRefIDInfoSize;
DWORD CallAttributeBitMask;

typedef struct SRTPKeyInfoStructure
{
    SRTPKeyInformation TransmissionSRTPInfo;
    SRTPKeyInformation ReceptionSRTPInfo;
} SRTPKeyInfoStructure;

typedef struct SRTPKeyInformation
{
    DWORDIsSRTPDataAvailable;
    DWORDSecureMediaIndicator; // CiscoSecurityIndicator
    DWORDMasterKeyOffset;
    DWORDMasterKeySize;
    DWORDMasterSaltOffset;
    DWORDMasterSaltSize;
    DWORDAlgorithmID; // CiscoSRTPAlgorithmIDs
    DWORDIsMKIPresent;
    DWORDKeyDerivationRate;
} SRTPKeyInformation;
enum CiscoSRTPAlgorithmIDs
{
    SRTP_NO_ENCRYPTION=0,
    SRTP_AES_128_COUNTER=1
};

enum CiscoSecurityIndicator
{
    SRTP_MEDIA_ENCRYPT_KEYS_AVAILABLE,
    SRTP_MEDIA_ENCRYPT_USER_NOT_AUTH,
    SRTP_MEDIA_ENCRYPT_KEYS_UNAVAILABLE,
    SRTP_MEDIA_NOT_ENCRYPTED
};

```

If `isSRTPInfoavailable` is set to `False`, rest of the information from `SRTPKeyInformation` must be ignored.

If `MasterKeySize` or `MasterSaltSize` is set to 0, then corresponding `MasterKeyOffset` or `MasterSaltOffset` must be ignored.

```

typedef struct DSCPValueInformation
{
    DWORD DSCPValueForAudioCalls;
}

typedef struct PartitionInformation
{
    DWORD CallerIDPartitionOffset;
    DWORD CallerIDPartitionSize;
    DWORD CalledIDPartitionOffset;
    DWORD CalledIDPartitionSize;
    DWORD ConnecetedIDPartitionOffset;
    DWORD ConnecetedIDPartitionSize;
    DWORD RedirectionIDPartitionOffset;
    DWORD RedirectionIDPartitionSize;
    DWORD RedirectingIDPartitionOffset;
    DWORD RedirectingIDPartitionSize;
} PartitionInformation;

Struct ExtendedCallInfo
{
    DWORD ExtendedCallReason ;
    DWORD CallerIDURLOffset;// CallPartySipURLInfo
    DWORD CallerIDURISize;
    DWORD CalledIDURLOffset;// CallPartySipURLInfo
    DWORD CalledIDURISize;
    DWORD ConnectedIDURIOffset;// CallPartySipURLInfo
    DWORD ConnectedIDURISize;
    DWORD RedirectionIDURIOffset;// CallPartySipURLInfo
    DWORD RedirectionIDURISize;
    DWORD RedirectingIDURIOffset;// CallPartySipURLInfo
    DWORD RedirectingIDURISize;
}

Struct CallPartySipURLInfo
{
    DWORD dwUserOffset; //sip user string
    DWORDdwUserSize;
    DWORDdwHostOffset; //host name string
    DWORDdwHostSize;
    DWORDdwPort;// integer port number
    DWORD dwTransportType; // SIP_TRANS_TYPE
        DWORD dwURLType;// SIP_URL_TYPE
}

enum {
    CTI_SIP_TRANSPORT_TCP=1,
    CTI_SIP_TRANSPORT_UDP,
    CTI_SIP_TRANSPORT_TLS
} SIP_TRANS_TYPE;
enum {
    CTI_NO_URL = 0,
    CTI_SIP_URL,
    CTI_TEL_URL
} SIP_URL_TYPE;

typedef struct CallAttributeInfo
{
    DWORD CallAttributeType;
    DWORD PartyDNOffset;
    DWORD PartyDNSize;
    DWORD PartyPartitionOffset;
}

```

```

DWORD PartyPartitionSize;
DWORD DeviceNameOffset;
DWORD DeviceNameSize;
DWORD OverallCallSecurityStatus;
}
typedef struct CallAttributeInfo_ExtA0
{
DWORD CallAttributeType;
DWORD PartyDNOffset;
DWORD PartyDNSize;
DWORD PartyPartitionOffset;
DWORD PartyPartitionSize;
DWORD DeviceNameOffset;
DWORD DeviceNameSize;
DWORD OverallCallSecurityStatus;
DWORD TransactionID;//Secure R & M
} CallAttributeInfo_ExtA0;

typedef struct CallAttributeInfo_ExtB0
{
CallAttributeInfo_ExtA0 attr_a0;
DWORD ActiveToneDirection;
} CallAttributeInfo_ExtB0;

typedef struct CCMCallHandleInformation
{
DWORD CCMCallID;
}

enum
{
    CallAttribute_Regular                = 0,
    CallAttribute_SilentMonitorCall,
    CallAttribute_SilentMonitorCall_Target,
    CallAttribute_RecordedCall,
    CallAttribute_WhisperCoachingCall,
    CallAttribute_WhisperCoachingCall_Target
} CallAttributeType
typedef struct CallSecurityStausInfo
{
DWORD CallSecurityStaus
} CallSecurityStausInfo
enum OverallCallSecurityStatus
{
OverallCallSecurityStatus_Unknown = 0,
OverallCallSecurityStatus_NotAuthenticated,
OverallCallSecurityStatus_Authenticated,
OverallCallSecurityStatus_Encrypted
};
typedef struct CPNInfo
{
    DWORD CallerPartyNumberType;//refer to CiscoNumberType
    DWORD CalledPartyNumberType;
    DWORD ConnectedIdNumberType;
    DWORD RedirectingPartyNumberType;
    DWORD RedirectionPartyNumberType;
        DWORD ModifiedCallingPartySize;
    DWORD ModifiedCallingPartyOffset;
    DWORD ModifiedCalledPartySize;
    DWORD ModifiedCalledPartyOffset;
    DWORD ModifiedConnectedIdSize;
    DWORD ModifiedConnectedIdOffset;
    DWORD ModifiedRedirectingPartySize;
    DWORD ModifiedRedirectingPartyOffset;

```

```

    DWORD ModifiedRedirectionPartySize;
    DWORD ModifiedRedirectionPartyOffset;
    DWORD GlobalizedCallingPartySize;
    DWORD GlobalizedCallingPartyOffset;
} CPNInfo;

enum CiscoNumberType {
    NumberType_Unknown = 0,          // UNKNOWN_NUMBER
    NumberType_International = 1,    // INTERNATIONAL_NUMBER
    NumberType_National = 2,        // NATIONAL_NUMBER
    NumberType_NetSpecificNum = 3,   // NET_SPECIFIC_NUMBER
    NumberType_Subscriber = 4,      // SUBSCRIBER_NUMBER
    NumberType_Abbreviated = 6      // ABBREVIATED_NUMBER
};

typedef struct Cisco_LineCallInfo_Ext000A0000
{
    ...
    //HuntList
    DWORD HuntPilotInfoOffset;//point to HuntPoiltInfo
    DWORD HuntPilotInfoSize;
    DWORD HuntPilotInfoCount;
    DWORD HuntPilotInfoElementFixedSize;
} CISCO_LINECALLINFO_EXT000A0000;

//HuntList
typedef struct HuntPilotInfo
{
    DWORD CallingPartyHuntPilotDNOffset;
    DWORD CallingPartyHuntPilotDNSize;
    DWORD CallingPartyHuntPilotPartitionOffset;
    DWORD CallingPartyHuntPilotPartitionSize;
    DWORD CalledPartyHuntPilotDNOffset;
    DWORD CalledPartyHuntPilotDNSize;
    DWORD CalledPartyHuntPilotPartitionOffset;
    DWORD CalledPartyHuntPilotPartitionSize;
    DWORD ConnectedPartyHuntPilotDNOffset;
    DWORD ConnectedPartyHuntPilotDNSize;
    DWORD ConnectedPartyHuntPilotPartitionOffset;
    DWORD ConnectedPartyHuntPilotPartitionSize;
} HuntPilotInfo;

typedef struct UniqueCallRefIDInfo
{
    DWORD UniqueCallRefIDOffset;
    DWORD UniqueCallRefIDSize;
} UniqueCallRefIDInfo;

typedef enum
{
    TSPCallAttribute_Normal = 0x00000000,
    TSPCallAttribute_IntercomOriginator = 0x00000001,
    TSPCallAttribute_IntercomTarget = 0x00000002,
    TSPCallAttribute_SilentMonitorCall = 0x00000004,
    TSPCallAttribute_SilentMonitorCall_Target = 0x00000008,
    TSPCallAttribute_RecordedCall = 0x00000010,
    TSPCallAttribute_ChaperoneCall = 0x00000020,
    TSPCallAttribute_CallForwardAllSet = 0x00000040,
    TSPCallAttribute_CallForwardAllClear = 0x00000080,
    TSPCallAttribute_WhisperMonitorCall = 0x00000100,
    TSPCallAttribute_WhisperMonitorCall_Target= 0x00000200,
    TSPCallAttribute_BIBCall = 0x00000400,
    TSPCallAttribute_ServerCall =0x00000800,
    TSPCallAttribute_SendMediaToBIB = 0x00001000
}

```

```
} CallAttributeBits
```

## Parameters

Parameter	Value
TapiCallerPartyUnicodeNameOffset TapiCallerPartyUnicodeNameSize	The size, in bytes, of the variably sized field that contains the Unicode Caller party identifier name information, and the offset, in bytes, from the beginning of the LINECALLINFO data structure
TapiCallerPartyLocale	The Unicode Caller party identifier name Locale information
TapiCalledPartyUnicodeNameOffset TapiCalledPartyUnicodeNameSize	The size, in bytes, of the variably sized field that contains the Unicode Called party identifier name information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
TapiCalledPartyLocale	The Unicode Called party identifier name locale information
TapiConnectedPartyUnicodeNameOffset TapiConnectedPartyUnicodeNameSize	The size, in bytes, of the variably sized field that contains the Unicode Connected party identifier name information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
TapiConnectedPartyLocale	The Unicode Connected party identifier name locale information
TapiRedirectionPartyUnicodeNameOffset TapiRedirectionPartyUnicodeNameSize	The size, in bytes, of the variably sized field that contains the Unicode Redirection party identifier name information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
TapiRedirectionPartyLocale	The Unicode Redirection party identifier name locale information
TapiRedirectingPartyUnicodeNameOffset TapiRedirectingPartyUnicodeNameSize	The size, in bytes, of the variably sized field that contains the Unicode Redirecting party identifier name information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
TapiRedirectingPartyLocale	The Unicode Redirecting party identifier name locale information
SRTPKeyInfoStructureOffset	Point to SRTPKeyInfoStructure
SRTPKeyInfoStructureSize	Total size of SRTP information
SRTPKeyInfoStructureElementCount	Number of SRTPKeyInfoStructure element
SRTPKeyInfoStructureElementFixedSize	Fixed size of SRTPKeyInfoStructure
SecureMediaIndicator	Indicates whether media is secure and whether application is authorized for key information
MasterKeyOffset MasterKeySize	The offset and size of SRTP MasterKey information
MasterSaltOffset MasterSaltSize	The offset and size of SRTP MasterSaltKey information

Parameter	Value
AlgorithmID	Specifies negotiated SRTP algorithm ID
IsMKIPresent	Indicates whether MKI is present
KeyDerivationRate	Provides the KeyDerivationRate
DSCPValueForAudioCalls	The DSCP value for Audio Calls
CallerIDPartitionOffset CallerIDPartitionSize	The size, in bytes, of the variably sized field that contains the Caller party identifier partition information and the offset, in bytes, from the beginning of LINECALLINFO data structure
CalledIDPartitionOffset CalledIDPartitionSize	The size, in bytes, of the variably sized field that contains the Called party identifier partition information and the offset, in bytes, from the beginning of LINECALLINFO data structure
ConnectedIDPartitionOffset ConnecetedIDPartitionSize	The size, in bytes, of the variably sized field that contains the Connected party identifier partition information and the offset, in bytes, from the beginning of LINECALLINFO data structure
RedirectionIDPartitionOffset RedirectionIDPartitionSize	The size, in bytes, of the variably sized field that contains the Redirection party identifier partition information, and the offset, in bytes, from the beginning of LINECALLINFO data structure
RedirectingIDPartitionOffset RedirectingIDPartitionSize	The size, in bytes, of the variably sized field that contains the Redirecting party identifier partition information and the offset, in bytes, from the beginning of LINECALLINFO data structure
ExtendedCallReason	<p>Presents all the last feature-related CTI Call reason code to the application as an extension to the standard reason codes that TAPI supports. This provides the feature-specific information per call. As phones that are running SIP are supported through CTI, new features can get introduced for phones that are running on SIP during releases.</p> <p><b>Note</b> Be aware that this field is not backward compatible and can change as changes or additions are made in the SIP phone support for a feature. Applications should implement some default behavior to handle any unknown reason codes that might be provided through this field.</p> <p>For Refer, the reason code specified is CtiCallReason_Refer.</p> <p>For Replaces, the reason code specified is CtiCallReason_Replaces.</p>
CallerIDURLOffset CallerIDURLSize	The size, in bytes, of the variably sized field that contains the Caller party identifier URL information and the offset, in bytes, from the beginning of LINECALLINFO data structure



Parameter	Value
CalledIDURLOffset CalledIDURLSize	The size, in bytes, of the variably sized field that contains the Called party identifier URL information and the offset, in bytes, from the beginning of LINECALLINFO data structure
ConnectedIDURLOffset ConnecetedIDURLSize	The size, in bytes, of the variably sized field that contains the Connected party identifier URL information and the offset, in bytes, from the beginning of LINECALLINFO data structure
RedirectionIDURLOffset RedirectionIDURLSize	The size, in bytes, of the variably sized field that contains the Redirection party identifier URL information and the offset, in bytes, from the beginning of LINECALLINFO data structure
RedirectingIDURLOffset RedirectingIDURLSize	The size, in bytes, of the variably sized field that contains the Redirecting party identifier URL information and the offset, in bytes, from the beginning of LINECALLINFO data structure
CallAttributeType	Identifies whether the following information (DN.Partition.DeviceName) is for a regular call, a monitoring call, a monitored call, or a recording call
PartyDNOffset, PartyDNSize,	The size, in bytes, of the variably sized field that contains the Monitoring/Monitored/Recorder party DN information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
PartyPartitionOffset PartyPartitionSize	The size, in bytes, of the variably sized field that contains the Monitoring/Monitored/Recorder party partition information and the offset, in bytes, from the beginning of the LINECALLINFO data structure
DeviceNameOffset DeviceNameSize	The size, in bytes, of the variably sized field that contains the Monitoring/Monitored/Recorder party device name and the offset, in bytes, from the beginning of the LINECALLINFO data structure
OverallCallSecurityStatus	The security status of the call for two-party calls and conference calls
CCMCallID	The Cisco Unified Communications Manager caller ID for each call leg
CallingPartyHuntPilotDNOffset CallingPartyHuntPilotDNSize	The size, in bytes, of the variably sized field containing the Hunt Pilot DN, and the offset, in bytes, from the beginning of LINECALLINFO data structure.
CallingPartyHuntPilotPartitionOffset CallingPartyHuntPilotPartitionSize	The size, in bytes, of the variably sized field containing the Hunt Pilot Partition, and the offset, in bytes, from the beginning of LINECALLINFO data structure.
CalledPartyHuntPilotDNOffset CalledPartyHuntPilotDNSize	The size, in bytes, of the variably sized field containing the Hunt Pilot DN, and the offset, in bytes, from the beginning of LINECALLINFO data structure.

Parameter	Value
CalledPartyHuntPilotPartitionOffset CalledPartyHuntPilotPartitionSize	The size, in bytes, of the variably sized field containing the Hunt Pilot Partition, and the offset, in bytes, from the beginning of LINECALLINFO data structure.
ConnectedPartyHuntPilotDNOffset ConnectedPartyHuntPilotDNSize	The size, in bytes, of the variably sized field containing the Hunt Pilot DN, and the offset, in bytes, from the beginning of LINECALLINFO data structure.
ConnectedPartyHuntPilotPartitionOffset ConnectedPartyHuntPilotPartitionSize	The size, in bytes, of the variably sized field containing the Hunt Pilot Partition, and the offset, in bytes, from the beginning of LINECALLINFO data structure.
IsChaperoneCall	This field specifies whether the call is a chaperone call or not.
UniqueCallRefIDInfoOffset UniqueCallRefIDInfoSize	The size, in bytes, of the variably sized field containing the Unique Call Reference identifier information, and the offset, in bytes, from the beginning of LINECALLINFO data structure.
CallType	This field contains information whether the call is generated as an usual outgoing call or due to CFwdAll softkey press on an on-hook device.

To indicate that partition information exists in the LINECALLINFO structure, the system fires a LINECALLINFOSTATE\_DEVSPECIFIC event. The bit map indicating the change is defined as the following:

```

SLDST_SRTP_INFO                0x00000001
SLDST_QOS_INFO                 0x00000002
SLDST_PARTITION_INFO           0x00000004
SLDST_EXTENDED_CALL_INFO       0x00000008
SLDST_CALL_ATTRIBUTE_INFO      0x00000010 //M&R
SLDST_CCM_CALL_ID              0x00000020 //M&R
SLDST_SECURITY_STATUS_INFO      0x00000040 //SecureConf
SLDST_NUMBER_TYPE_CHANGED      0x00000080 //CPN
SLDST_GLOBALIZED_CALLING_PARTY_CHANGED 0x00000100 //CPN
SLDST_FAR_END_IP_ADDRESS_CHANGED 0x00000200 //IPv6 new
SLDST_UNIQUE_CALL_REF_ID_INFO  0x00000400

LINEDEVSPECIFIC
{
    hDevice = hcall //call handle for which the info has changed.
    dwParam1 = SLDSMT_LINECALLINFO_DEVSPECIFICDATA //indicates DevSpecific portion's changed
    dwParam2 = SLDST_SRTP_INFO | SLDST_QOS_INFO | SLDST_PARTITION_INFO |
SLDST_EXTENDED_CALL_INFO | SLDST_CALL_ATTRIBUTE_INFO | SLDST_CCM_CALLID |
SLDST_CALL_SECURITY_STATUS | SLDST_NUMBER_TYPE_CHANGED |
SLDST_GLOBALIZED_CALLING_PARTY_CHANGED | SLDST_FAR_END_IP_ADDRESS_CHANGED |
SLDST_UNIQUE_CALL_REF_ID_INFO
    dwParam3 = ...
    dwParam3 will be security indicator if dwParam2 has bit set for SLDST_SRTP_INFO
}

```

## LINEDEVSTATUS

Cisco TSP implements several line device-specific extensions and uses the DevSpecific (dwDevSpecificSize and dwDevSpecificOffset) variably sized area of the LINEDEVSTATUS data structure for those extensions. Cisco TSP defines the DevSpecific area layout in the Cisco\_LineDevStatus\_Ext structure in the CiscoLineDevSpecificMsg.h header file. The extension version in which the data was introduced provides basis for how the data in that structure is organized.

```
// LINEDEVSTATUS Dev Specific extension //
typedef struct Cisco_LineDevStatus_Ext
{
    Cisco_LineDevStatus_Ext00060000 ext60;
    Cisco_LineDevStatus_Ext00070000 ext70;
    Cisco_LineDevStatus_Ext00080000 ext80;
} CISCO_LINEDEVSTATUS_EXT;
```

For a specific line device, the extension area will include a portion of this structure, starting from the beginning and up to the extension version that an application negotiated.

### Detail

The individual extension version substructure definitions follow:

```
// LINEDEVSTATUS 00060000 extention //
typedef struct Cisco_LineDevStatus_Ext00060000
{
    DWORD dwSupportedEncoding;
} CISCO_LINEDEVSTATUS_EXT00060000;
// LINEDEVSTATUS 00070000 extention //
typedef struct Cisco_LineDevStatus_Ext00070000
{
    char lpszAlternateScript[MAX_ALTERNATE_SCRIPT_SIZE];
    // An empty string means there is no alternate script configured
    // or the phone does not support alternate scripts
} CISCO_LINEDEVSTATUS_EXT00070000;
// LINEDEVSTATUS 00080000 extention //
// Status extention 00080000 Data Structure//
typedef struct Cisco_LineDevStatus_Ext00080000
{
    CISCOLINEDEVSTATUS_DONOTDISTURB doNotDisturbStatus;
} CISCO_LINEDEVSTATUS_EXT00080000;
typedef struct CiscoLineDevStatus_DoNotDisturb
{
    DWORD m_LineDevStatus_DoNotDisturbOption;
    DWORD m_LineDevStatus_DoNotDisturbStatus;
} CISCOLINEDEVSTATUS_DONOTDISTURB;
```

You can find additional information on the DevSpecific structure layout and data in the CiscoLineDevSpecificMsg.h header file.

The CiscoLineDevStatus\_DoNotDisturb structure belongs to the LINEDEVSTATUS\_DEV\_SPECIFIC\_DATA structure and gets used to reflect the current state of the Do Not Disturb feature.

### Parameters

DWORD dwSupportEncoding

This parameter indicates the Support Encoding for the Unicode Party names that are being sent in device-specific extension of the LINECALLINFO structure.

The typical values could be

```
enum {
    UnknownEncoding = 0, // Unknown encoding
    NotApplicableEncoding = 1, // Encoding not applicable to this device
    AsciiEncoding = 2,      // ASCII encoding
    Ucs2UnicodeEncoding = 3 // UCS-2 Unicode encoding
}
```




---

**Note** Be aware that the `dwSupportedEncoding` extension is only available if extension version 0x00060000 or higher is negotiated.

---

### LPCSTR lpszAlternateScript

This parameter specifies the alternate script that the device supports. An empty string indicates the device does not support or is not configured with an alternate script.

The only supported script in this release is "Kanji" for the Japanese locale.

### m\_LineDevStatus\_DoNotDisturbOption

This field contains DND option that is configured for the device and can comprise one of the following enum values:

```
enum CiscoDoNotDisturbOption {
    DoNotDisturbOption_NONE = 0,
    DoNotDisturbOption_RINGEROFF = 1,
    DoNotDisturbOption_REJECT = 2
};
```

`m_LineDevStatus_DoNotDisturbStatus` field contains current DND status on the device and can be one of the following enum values:

```
enum CiscoDoNotDisturbStatus {
    DoNotDisturbStatus_UNKNOWN = 0,
    DoNotDisturbStatus_ENABLED = 1,
    DoNotDisturbStatus_DISABLED = 2
};
```




---

**Note** Be aware that this extension is only available if extension version 8.0 (0x00080000) or higher is negotiated.

---

## CCiscoLineDevSpecific

This section provides information on how to perform Cisco Unified TAPI specific functions with the `CCiscoLineDevSpecific` class, which represents the parent class to all the following classes. It comprises a virtual class and is provided here for informational purposes.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificMsgWaiting
|
+--CCiscoLineDevSpecificMsgWaitingDirn
|
+--CCiscoLineDevSpecificUserControlRTPStream
|
```

```

+--CCiscoLineDevSpecificSetStatusMsgs
|
+--CCiscoLineDevSpecificSwapHoldSetupTransfer
|
+--CCiscoLineDevSpecificRedirectResetOrigCalled
|
+--CCiscoLineDevSpecificRedirectSetOrigCalled
|
+--CCiscoLineDevSpecificPortRegistrationPerCall
|
+--CCiscoLineDevSpecificSetRTPParamsForCall
|
+--CCiscoLineDevSpecificJoin
|
+--CCiscoLineDevSpecificRedirectFACCMC
|
+--CCiscoLineDevSpecificBlindTransferFACCMC
|
+--CCiscoLineDevSpecificCTIPortThirdPartyMonitor
|
+--CCiscoLineDevSpecificUserSetSRTPAlgorithmID
|
+--CCiscoLineDevSpecificSendLineOpen
|
+--CCiscoLineDevSpecificAcquire
|
+--CCiscoLineDevSpecificDeacquire
|
+--CCiscoLineDevSpecificStartCallMonitoring
|
+--CCiscoLineDevSpecificStartCallRecording
|
+--CCiscoLineDevSpecificStopCallRecording
|
+--CCiscoLineSetIntercomSpeeddial
|
+--CCiscoLineIntercomTalkback
|
+--CciscoSetupTransferWithoutMedia
|
+--CCiscoLineDevSpecificSetMsgSummary
|
+--CCiscoLineDevSpecificSetMsgSummaryDirn
|
+--CCiscoLineDevSpecificSetRTPParamsForCallIPv6
|
+--CCiscoLineDevSpecificSetIPv6AddressAndMode
|
+--CCiscoLineDevSpecificDirectTransfer
|
+--CCiscoLineDevSpecificRegisterCallPickupGroupForNotification
|
+--CCiscoLineDevSpecificUnRegisterCallPickupGroupForNotification
|
+--CCiscoLineDevSpecificCallPickupRequest
|
+--CCiscoLineDevSpecificPlaytone
|
+--CCiscoLineDevSpecificStartSendMediaToBIBRequest
|
+--CCiscoLineDevSpecificStopSendMediaToBIBRequest
|
+--CCiscoLineDevSpecificMonitoringUpdateMode
|

```

```

+--CCiscoLineDevSpecificEnableFeatureSupport
|
+--CCiscoLineRedirectWithFeaturePriority

```

## Header File

The file `CiscoLineDevSpecific.h` contains the constant, structure, and class definition for the Cisco line device-specific classes.

## Class Detail

```

class CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecific(DWORD msgType);
    virtual ~CCiscoLineDevSpecific();
    DWORD GetMsgType(void) const {return m_MsgType;}
    void* lpParams() {return &m_MsgType;}
    virtual DWORD dwSize() = 0;
private:
    DWORD m_MsgType;
};

```

## Functions

`lpParams()`

You can use function to obtain the pointer to the parameter block.

`dwSize()`

Function will give the size of the parameter block area.

## Parameter

`m_MsgType`

Specifies the type of message.

## Subclasses

Each subclass of `CCiscoLineDevSpecific` includes a different value that is assigned to the parameter `m_MsgType`. If you are using C instead of C++, this represents the first parameter in the structure.

## Enumeration

The `CiscoLineDevSpecificType` enumeration provides valid message identifiers.

```

enum CiscoLineDevSpecificType
{
    SLDST_MSG_WAITING = 1,
    SLDST_MSG_WAITING_DIRN,
    SLDST_USER_CTRL_OF_RTP_STREAM,
    SLDST_SET_STATUS_MESSAGES,
    SLDST_NUM_TYPE,
    SLDST_SWAP_HOLD_SETUP_TRANSFER, // Not Supported in CiscoTSP 3.4 and Beyond
    SLDST_REDIRECT_RESET_ORIG_CALLED,

```

```

SLDST_REDIRECT_SET_ORIG_CALLED,
SLDST_USER_RECEIVE_RTP_INFO,
SLDST_USER_SET_RTP_INFO,
SLDST_JOIN,
SLDST_REDIRECT_FAC_CMC,
SLDST_BLIND_TRANSFER_FAC_CMC,
SLDST_CTI_PORT_THIRD_PARTY_MONITOR,
SLDST_ACQUIRE,
SLDST_DE_ACQUIRE,
SLDST_USER_SET_SRTP_ALGORITHM_ID,
SLDST_SEND_LINE_OPEN,
SLDST_START_CALL_MONITORING,
SLDST_START_CALL_RECORDING,
SLDST_STOP_CALL_RECORDING,
SLDST_LINE_SET_INTERCOM_SPEEDDIAL,
SLDST_LINE_INTERCOM_TALKBACK,
SLDST_REDIRECT_WITH_FEATURE_PRIORITY,
SLDST_USER_SET_RTP_INFO_IPv6,
SLDST_USER_SET_IPv6_ADDRESS_AND_MODE,
SLDST_SETUP_TRANSFER_WITHOUT_MEDIA,
SLDST_DIRECT_TRANSFER,
SLDST_MSG_SUMMARY,
SLDST_MSG_SUMMARY_DIRN,
SLDST_CALLPICKUP_GROUP_REGISTER,
SLDST_CALLPICKUP_GROUP_UNREGISTER,
SLDST_CALLPICKUP_CALL,
SLDST_PLAY_TONE,
SLDST_START_SEND_MEDIA_TO_BIB,
SLDST_STOP_SEND_MEDIA_TO_BIB,
SLDST_UPDATE_MONITOR_MODE,
SLDST_ENABLE_FEATURE_SUPPORT
};

```

## Message Waiting

The `CCiscoLineDevSpecificMsgWaiting` class turns the message waiting lamp on or off for the line that the `hLine` parameter specifies.



### Note

This extension does not require an extension version to be negotiated.

```

CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificMsgWaiting

```

## Class Detail

```

class CCiscoLineDevSpecificMsgWaiting : public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificMsgWaiting() : CCiscoLineDevSpecific(SLDST_MSG_WAITING) {}
    virtual ~CCiscoLineDevSpecificMsgWaiting() {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    DWORD m_BlinkRate;
};

```

## Parameters

DWORD m\_MsgType

Equals SLDST\_MSG\_WAITING.

DWORD m\_BlinkRate

Any supported PHONELAMPMODE\_ constants that are specified in the phoneSetLamp() function.



### Note

---

Cisco Unified IP Phone 7900 Series supports only PHONELAMPMODE\_OFF and PHONELAMPMODE\_STEADY

---

## Message Waiting Dirn

The CCiscoLineDevSpecificMsgWaitingDirn class turns the message waiting lamp on or off for the line that a parameter specifies and remains independent of the hLine parameter.



### Note

---

This extension does not require an extension version to be negotiated.

---

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificMsgWaitingDirn
```

## Class Detail

```
class CCiscoLineDevSpecificMsgWaitingDirn : public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificMsgWaitingDirn() :
        CCiscoLineDevSpecific(SLDST_MSG_WAITING_DIRN) {}
    virtual ~CCiscoLineDevSpecificMsgWaitingDirn() {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    DWORD m_BlinkRate;
    char m_Dirn[25];
};
```

## Parameters

DWORD m\_MsgType

Specifies SLDST\_MSG\_WAITING\_DIRN.

DWORD m\_BlinkRate

As in the CCiscoLineDevSpecificMsgWaiting message.



### Note

---

Cisco Unified IP Phone 7900 Series supports only PHONELAMPMODE\_OFF and PHONELAMPMODE\_STEADY

---

char m\_Dirn[25]

The directory number for which the message waiting lamp should be set.



## Message Summary

Use the `CCiscoLineDevSpecificSetMsgSummary` class to turn the message waiting lamp on or off as well as to provide voice and fax message counts for the line specified by the `hLine` parameter.



### Note

Be aware that this extension does not require an extension version to be negotiated.

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificSetMsgSummary
```

## Class Detail

```
class CCiscoLineDevSpecificSetMsgSummary : public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificSetMsgSummary() : CCiscoLineDevSpecific(SLDST_MSG_SUMMARY){}
    virtual ~CCiscoLineDevSpecificSetMsgSummary() {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    DWORD m_BlinkRate;
    MSG_SUMMARY m_MessageSummary;
};
```

## Parameters

`DWORD m_MsgType`

equals `SLDST_MSG_SUMMARY`.

`DWORD m_BlinkRate`

is any supported `PHONELAMPMODE_` constants specified in the `phoneSetLamp()` function.

`MSG_SUMMARY m_MessageSummary`

A data structure with the following format:

```
typedef struct {
DWORD m_voiceCounts; // indicates if new voice counts are
                        // provided. True=counts will be displayed
                        // on supported phones.
DWORD m_totalNewVoiceMsgs; // specifies the total number of new
                        // voice messages. This number includes all
                        // the high and normal priority voice
                        // messages that are new.
DWORD m_totalOldVoiceMsgs; // specifies the total number of old
                        // voice messages. This number includes all
                        // high and normal priority voice messages
                        // that are old.
DWORD m_highPriorityVoiceCounts; // indicates if old voice
                        // counts are provided. True=counts will be
                        // displayed on supported phones.
DWORD m_newHighPriorityVoiceMsgs; //specifies the number of new
                        // high priority voice messages.
DWORD m_oldHighPriorityVoiceMsgs; //specifies the number of old
                        // high priority voice messages.
DWORD m_faxCounts; // indicates if new fax counts are
                        // provided. True=counts will be displayed
                        // on supported phones.
DWORD m_totalNewFaxMsgs; // specifies the total number of new
```

```

// fax messages. This number includes all
// the high and normal priority fax
// messages that are new.
DWORD m_totalOldFaxMsgs; // specifies the total number of old
// fax messages. This number includes all
// high and normal priority fax messages
// that are old.
DWORD m_highPriorityFaxCounts; // indicates if old fax counts
// are provided. True=counts will be
// displayed on supported phones.
DWORD m_newHighPriorityFaxMsgs; // specifies the number of new
// high priority fax messages.
DWORD m_oldHighPriorityFaxMsgs; // specifies the number of old
// high priority fax messages.
} MSG_SUMMARY;

```

## Message Summary Dirn

Use the `CCiscoLineDevSpecificSetMsgSummaryDirn` class to turn the message waiting lamp on or off and to provide voice and fax message counts for the line specified by a parameter and is independent of the `hLine` parameter.



### Note

Be aware that this extension does not require an extension version to be negotiated.

```

CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificSetMsgSummaryDirn

```

## Class Detail

```

class CCiscoLineDevSpecificSetMsgSummaryDirn : public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificSetMsgSummaryDirn() : CCiscoLineDevSpecific(SLDST_MSG_SUMMARY_DIRN)
    {}
    virtual ~CCiscoLineDevSpecificSetMsgSummaryDirn() {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    DWORD m_BlinkRate;
    char m_Dirn[25];
    MSG_SUMMARY m_MessageSummary;
};

```

## Parameters

DWORD `m_MsgType`

equals `SLDST_MSG_SUMMARY_DIRN`.

DWORD `m_BlinkRate`

is as in the `CCiscoLineDevSpecificSetMsgSummary` message.

char `m_Dirn[25]`

is the directory number for which the message waiting lamp should be set.

MSG\_SUMMARY `m_MessageSummary`

A data structure with the following format:

```
typedef struct {
    DWORD m_voiceCounts; // indicates if new voice counts are
                        // provided. True=counts will be displayed
                        // on supported phones.
    DWORD m_totalNewVoiceMsgs; // specifies the total number of new
                        // voice messages. This number includes all
                        // the high and normal priority voice
                        // messages that are new.
    DWORD m_totalOldVoiceMsgs; // specifies the total number of old
                        // voice messages. This number includes all
                        // high and normal priority voice messages
                        // that are old.
    DWORD m_highPriorityVoiceCounts; // indicates if old voice
                        // counts are provided. True=counts will be
                        // displayed on supported phones.
    DWORD m_newHighPriorityVoiceMsgs; //specifies the number of new
                        // high priority voice messages.
    DWORD m_oldHighPriorityVoiceMsgs; //specifies the number of old
                        // high priority voice messages.
    DWORD m_faxCounts; // indicates if new fax counts are
                        // provided. True=counts will be displayed
                        // on supported phones.
    DWORD m_totalNewFaxMsgs; // specifies the total number of new
                        // fax messages. This number includes all
                        // the high and normal priority fax
                        // messages that are new.
    DWORD m_totalOldFaxMsgs; // specifies the total number of old
                        // fax messages. This number includes all
                        // high and normal priority fax messages
                        // that are old.
    DWORD m_highPriorityFaxCounts; // indicates if old fax counts
                        // are provided. True=counts will be
                        // displayed on supported phones.
    DWORD m_newHighPriorityFaxMsgs; // specifies the number of new
                        // high priority fax messages.
    DWORD m_oldHighPriorityFaxMsgs; // specifies the number of old
                        // high priority fax messages.
} MSG_SUMMARY;
```

## Audio Stream Control

The `CCiscoLineDevSpecificUserControlRTPStream` class controls the audio stream of a line. To use this class you must call the `lineNegotiateExtVersion` API before opening the line. When `lineNegotiateExtVersion` is called ensure the highest bit is set on both the `dwExtLowVersion` and `dwExtHighVersion` parameters. This causes the call to `lineOpen` to behave differently. The line does not actually open, but waits for a `lineDevSpecific` call to complete the open with more information. The `CCiscoLineDevSpecificUserControlRTPStream` class provides the extra information that is required.

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificUserControlRTPStream
```

### Procedure

- 
- Step 1** Call `lineNegotiateExtVersion` for the `deviceID` of the line that is to be opened (OR `0x80000000` with the `dwExtLowVersion` and `dwExtHighVersion` parameters).
  - Step 2** Call `lineOpen` for the `deviceID` of the line that is to be opened.
  - Step 3** Call `lineDevSpecific` with a `CCiscoLineDevSpecificUserControlRTPStream` message in the `lpParams` parameter.
- 

## Class Detail

```
class CCiscoLineDevSpecificUserControlRTPStream : public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificUserControlRTPStream() :
        CCiscoLineDevSpecific(SLDST_USER_CTRL_OF_RTP_STREAM),
        m_ReceiveIP(-1),
        m_ReceivePort(-1),
        m_NumAffectedDevices(0)
        {
            memset(m_AffectedDeviceID, 0, sizeof(m_AffectedDeviceID));
        }
    virtual ~CCiscoLineDevSpecificUserControlRTPStream() {}
    DWORD m_ReceiveIP; // UDP audio reception IP
    DWORD m_ReceivePort; // UDP audio reception port
    DWORD m_NumAffectedDevices;
    DWORD m_AffectedDeviceID[10];
    DWORD m_MediaCapCount;
    MEDIA_CAPS m_MediaCaps;
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
};
```

## Parameters

DWORD `m_MsgType`

Equals `SLDST_USER_CTRL_OF_RTP_STREAM`

DWORD `m_ReceiveIP`:

The RTP audio reception IP address in network byte order

DWORD `m_ReceivePort`:

The RTP audio reception port in network byte order

DWORD m\_NumAffectedDevices:

The TSP returns this value. It contains the number of deviceIDs in the m\_AffectedDeviceID array that are valid. Any device with multiple directory numbers that are assigned to it will have multiple TAPI lines, one per directory number.

DWORD m\_AffectedDeviceID[10]:

The TSP returns this value. It contains the list of deviceIDs for any device that is affected by this call. Do not call lineDevSpecific for any other device in this list.

DWORD m\_mediaCapCount

The number of codecs that are supported for this line.

MEDIA\_CAPS m\_MediaCaps -

A data structure with the following format:

```
typedef struct {
    DWORD MediaPayload;
    DWORD MaxFramesPerPacket;
    DWORD G723BitRate;
} MEDIA_CAPS[MAX_MEDIA_CAPS_PER_DEVICE];
```

This data structure defines each codec that is supported on a line. The limit specifies 18. The following description shows each member in the MEDIA\_CAPS data structure:

MediaPayload specifies an enumerated integer that contains one of the following values:

```
enum
{
    Media_Payload_G711Alaw64k = 2,
    Media_Payload_G711Alaw56k = 3, // "restricted"
    Media_Payload_G711Ulaw64k = 4,
    Media_Payload_G711Ulaw56k = 5, // "restricted"
    Media_Payload_G722_64k = 6,
    Media_Payload_G722_56k = 7,
    Media_Payload_G722_48k = 8,
    Media_Payload_G7231 = 9,
    Media_Payload_G728 = 10,
    Media_Payload_G729 = 11,
    Media_Payload_G729AnnexA = 12,
    Media_Payload_G729AnnexB = 15,
    Media_Payload_G729AnnexAwAnnexB = 16,
    Media_Payload_GSM_Full_Rate = 18,
    Media_Payload_GSM_Half_Rate = 19,
    Media_Payload_GSM_Enhanced_Full_Rate = 20,
    Media_Payload_Wide_Band_256k = 25,
    Media_Payload_Data64 = 32,
    Media_Payload_Data56 = 33,
    Media_Payload_GSM = 80,
    Media_Payload_G726_32K = 82,
    Media_Payload_G726_24K = 83,
    Media_Payload_G726_16K = 84,
    // Media_Payload_G729_B = 85,
    // Media_Payload_G729_B_LOW_COMPLEXITY = 86,
} Media_PayloadType;
```

Read MaxFramesPerPacket as MaxPacketSize. It specifies a 16-bit integer that indicates the maximum desired RTP packet size in milliseconds. Typically, this value gets set to 20.

G723BitRate specifies a 6-byte field that contains either the G.723.1 information bit rate, or it gets ignored. The following list provides values for the G.723.1 field values:

```
enum
{
    Media_G723BRate_5_3 = 1, //5.3Kbps
    Media_G723BRate_6_4 = 2 //6.4Kbps
} Media_G723BitRate;
```

## Set Status Messages

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificSetStatusMsgs
```

## Description

Use the `CCiscoLineDevSpecificSetStatusMsgs` class to turn on or off the status messages for the line that the `hLine` parameter specifies. The Cisco Unified TSP supports the following flags:

- `DEVSPECIFIC_MEDIA_STREAM`—Setting this flag on a line turns on the reporting of media streaming messages for that line. Clearing this flag turns off the reporting of media streaming messages for that line.
- `DEVSPECIFIC_CALL_TONE_CHANGED`—Setting this flag on a line turns on the reporting of call tone changed events for that line. Clearing this flag turns off the reporting of call tone changed events for that line.
- `DEVSPECIFIC_SILENT_MONITORING_TERMINATED`—Setting this flag on a line turns on the reporting of Monitoring Session Terminated Event messages for that line. Clearing this flag turns off the reporting of Monitoring Session Terminated Event Messages for that line.
- `DEVSPECIFIC_GET_IP_PORT`—Setting this flag on a line turns on the reporting of Get IP and Port Notification Event messages for that line. Clearing this flag turns off the reporting of Get IP and Port Notification Event Messages for that line.
- `DEVSPECIFIC_HOLD_REVERSION`—Setting this flag on a line causes the application to receive a `LINE_DEVSPECIFIC(dwParam1 = SLDSMT_HOLD_REVERSION)` when a hold reversion happens on a held call. Clearing this flag on a line turns off the reporting of the `LINE_DEVSPECIFIC(dwParam1 = SLDSMT_HOLD_REVERSION)` event.
- `DEVSPECIFIC_IDLE_TRANSFER_REASON`—Setting this flag on a line causes the reason to be reported as `LINECALLREASON_TRANSFER` when calls go to the `LINECALLSTATE_IDLE` state after a transfer is completed at the transfer controller. Clearing this flag on a line causes the reason to be reported as `LINECALLREASON_DIRECT` when calls go to the `LINECALLSTATE_IDLE` state after a transfer is completed at the transfer controller.
- `DEVSPECIFIC_SPEEDDIAL_CHANGED`—Setting this flag on a line causes a `LINE_DEVSPECIFIC(dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED, dwParam2 = LPCT_INTERCOM_LINE, and dwParam3 = CiscoIntercomLineChangeResult)` to be fired to the application when there is a change in the database or the application overwrites the speed dial setting. Clearing this flag turns off the reporting of the `LINE_DEVSPECIFIC(dwParam1 = SLDSMT_LINE_PROPERTY_CHANGED, dwParam2 = LPCT_INTERCOM_LINE, and dwParam3 = CiscoIntercomLineChangeResult)` event.
- `DEVSPECIFIC_DONOTDISTURB_CHANGED`—Setting this flag on a line causes a `LINE_DEVSPECIFICFEATURE(dwParam1 = PHONEBUTTONFUNCTION_DONOTDISTURB, dwParam2 = typeOfChange, and dwParam3 = currentValue)` to be fired to the application when there

is a change in the DND configuration or status. Clearing this flag turns off the reporting of the LINE\_DEVSPECIFICFEATURE(dwParam1 = PHONEBUTTONFUNCTION\_DONOTDISTURB, dwParam2 = typeOfChange, and dwParam3 = currentValue) event.

- **DEVSPECIFIC\_DISPLAYABLE\_ADDRESS**—Setting this flag on a line causes the DisplayableAddress field in LINECALLINFO to be filled with the latest called partyDN/ASCCI name/Unicode name/Partition (separated by ":"). Clearing this flag causes the DisplayableAddress field in LINECALLINFO to be empty.
- **DEVSPECIFIC\_DEVICE\_STATE**—Setting this flag gets the accumulative state of all the lines on the device and with the state being fired to the application using the LINE\_DEVSPECIFIC(dwParam1 = SLDSMT\_DEVICE\_STATE, dwParam2 = State) events. Clearing this flag turns off the reporting of the accumulative state of all the lines on the device.

The DEVSPECIFIC\_DEVICE\_STATE state is defined as:

```
enum lineDeviceState
{
    lineDeviceState_UNKNOWN = 0,
    lineDeviceState_ACTIVE = 1,
    lineDeviceState_ALERTING = 2,
    lineDeviceState_HELD = 3,
    lineDeviceState_WHISPER = 4,
    lineDeviceState_IDLE = 5
};
```

- **DEVSPECIFIC\_PARK\_MONITORING**—Setting this flag on a line causes the Park Monitoring events to be fired to the application. Clearing this flag turns off the reporting of the Park Monitoring events. For more information, see [Park Monitoring, page 3-36](#).
- **DEVSPECIFIC\_OTHER\_DEVICE\_STATE\_NOTIFY**—Setting this flag on a line notifies the application about the non-opened device state changes. Clearing this flag turns off the reporting of the other non-opened device state changes. For more information, see [Other-Device State Notification, page 3-35](#).



#### Note

This extension only applies if extension version 0x00020001 or higher is negotiated.

## Class Detail

```
class CCiscoLineDevSpecificSetStatusMsgs : public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificSetStatusMsgs() :
    CCiscoLineDevSpecific(SLDST_SET_STATUS_MESSAGES) {}
    virtual ~CCiscoLineDevSpecificSetStatusMsgs() {}
    DWORD m_DevSpecificStatusMsgsFlag;
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
};
```

## Parameters

DWORD m\_MsgType

Equals SLDST\_SET\_STATUS\_MESSAGES.

DWORD m\_DevSpecificStatusMsgsFlag

Identifies which status changes cause a LINE\_DEVSPECIFIC message to be sent to the application.

The supported values follow:

```

#define DEVSPECIFIC_MEDIA_STREAM0x00000001
#define DEVSPECIFIC_CALL_TONE_CHANGED0x00000002
#define CALL_DEVSPECIFIC_RTP_EVENTS0x00000003
#define DEVSPECIFIC_IDLE_TRANSFER_REASON 0x00000004
#define DEVSPECIFIC_HOLD_REVERSION 0x00000008
#define DEVSPECIFIC_SPEEDDIAL_CHANGED0x00000010
#define DEVSPECIFIC_DONOTDISTURB_CHANGED0x00000020
#define DEVSPECIFIC_DISPLAYABLE_ADDRESS0x00000040
#define DEVSPECIFIC_PARK_MONITORING0x00000080
#define DEVSPECIFIC_DEVICE_STATE0x00000100
#define DEVSPECIFIC_SILENT_MONITORING_TERMINATED0x00000200
#define DEVSPECIFIC_OTHER_DEVICE_STATE_NOTIFY0x00000400
#define DEVSPECIFIC_GET_IP_PORT0x00000800

```

## Swap-Hold/SetupTransfer



### Note

---

Cisco Unified TSP 4.0 and later do not support this.

---

The `CCiscoLineDevSpecificSwapHoldSetupTransfer` class gets used to perform a `SetupTransfer` between a call that is in `CONNECTED` state and a call that is in the `ONHOLD` state. This function changes the state of the connected call to `ONHOLDPENDTRANSFER` state and the `ONHOLD` call to `CONNECTED` state. This allows a `CompleteTransfer` to be performed on the two calls. In Cisco Unified TSP 4.0 and later, the TSP allows applications to use `lineCompleteTransfer()` to transfer the calls without having to use the `CCiscoLineDevSpecificSwapHoldSetupTransfer` function. Therefore, this function returns `LINEERR_OPERATIONUNAVAIL` in Cisco Unified TSP 4.0 and beyond.

```

CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificSwapHoldSetupTransfer

```



### Note

---

This extension only applies if extension version `0x00020002` or higher is negotiated.

---

## Class Details

```

class CCiscoLineDevSpecificSwapHoldSetupTransfer : public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificSwapHoldSetupTransfer() :
CCiscoLineDevSpecific(SLDST_SWAP_HOLD_SETUP_TRANSFER) {}
    virtual ~CCiscoLineDevSpecificSwapHoldSetupTransfer() {}
    DWORD heldCallID;
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;} // subtract out the
virtual function table pointer
};

```

## Parameters

`DWORD m_MsgType`

Equals `SLDST_SWAP_HOLD_SETUP_TRANSFER`.

`DWORD heldCallID`

Equals the callid of the held call that is returned in `dwCallID` of `LPLINECALLINFO`.



HCALL hCall (in lineDevSpecific parameter list)

Equals the handle of the connected call.

## Redirect Reset Original Called ID

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificRedirectResetOrigCalled
```

### Description

The `CCiscoLineDevSpecificRedirectResetOrigCalled` class redirects a call to another party while it resets the original called ID of the call to the destination of the redirect.



#### Note

---

This extension only applies if extension version 0x00020003 or higher is negotiated.

---

### Class Details

```
class CCiscoLineDevSpecificRedirectResetOrigCalled: public CCiscoLineDevSpecific
{
    public:
        CCiscoLineDevSpecificRedirectResetOrigCalled:
CCiscoLineDevSpecific(SLDST_REDIRECT_RESET_ORIG_CALLED) {}
        virtual ~CCiscoLineDevSpecificRedirectResetOrigCalled{}
        char m_DestDirn[25]; //redirect destination address
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;} // subtract out the
virtual function table pointer
};
```

### Parameters

DWORD m\_MsgType

Equals `SLDST_REDIRECT_RESET_ORIG_CALLED`.

DWORD m\_DestDirn

Equals the destination address where the call needs to be redirected.

HCALL hCall (In lineDevSpecific parameter list)

Equals the handle of the connected call.

## Port Registration per Call

The `CCiscoLineDevSpecificPortRegistrationPerCall` class registers the CTI Port for the RTP parameters on a per-call basis. With this request, the application receives the new `lineDevSpecific` event that requests that it needs to set the RTP parameters for the call.

To use this class, ensure the `lineNegotiateExtVersion` API is called before opening the line. When calling `lineNegotiateExtVersion`, ensure the highest bit is set on both the `dwExtLowVersion` and `dwExtHighVersion` parameters.

This causes the call to `lineOpen` to behave differently. The line does not actually open, but waits for a `lineDevSpecific` call to complete the open with more information. The extra information required is provided in the `CciscoLineDevSpecificPortRegistrationPerCall` class.

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificPortRegistrationPerCall
```

### Procedure

- 
- Step 1** Call `lineNegotiateExtVersion` for the `deviceID` of the line that is to be opened (or `0x80000000` with the `dwExtLowVersion` and `dwExtHighVersion` parameters)
  - Step 2** Call `lineOpen` for the `deviceID` of the line that is to be opened.
  - Step 3** Call `lineDevSpecific` with a `CciscoLineDevSpecificPortRegistrationPerCall` message in the `lpParams` parameter.
- 



#### Note

This extension is only available if the extension version `0x00040000` or higher gets negotiated.

---

## Class Details

```
class CCiscoLineDevSpecificPortRegistrationPerCall: public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificPortRegistrationPerCall () :
        CCiscoLineDevSpecific(SLDST_USER_RECEIVE_RTP_INFO),
        m_RecieveIP(-1), m_RecievePort(-1), m_NumAffectedDevices(0)
    {
        memset((char*)m_AffectedDeviceID, 0, sizeof(m_AffectedDeviceID));
    }

    virtual ~ CCiscoLineDevSpecificPortRegistrationPerCall () {}
    DWORD m_NumAffectedDevices;
    DWORD m_AffectedDeviceID[10];
    DWORD m_MediaCapCount;
    MEDIA_CAPSm_MediaCaps;
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    // subtract out the virtual function table pointer
};
```

## Parameters

`DWORD m_MsgType`  
 Equals `SLDST_USER_RECEIVE_RTP_INFO`

DWORD m\_NumAffectedDevices:

TSP returns this value. It contains the number of deviceIDs in the m\_AffectedDeviceID array that are valid. Any device with multiple directory numbers that are assigned to it will have multiple TAPI lines, one per directory number.

DWORD m\_AffectedDeviceID[10]:

TSP returns this value. It contains the list of deviceIDs for any device that is affected by this call. Do not call lineDevSpecific for any other device in this list.

DWORD m\_mediaCapCount

The number of codecs that are supported for this line.

MEDIA\_CAPS m\_MediaCaps -

A data structure with the following format:

```
typedef struct {
    DWORD MediaPayload;
    DWORD MaxFramesPerPacket;
    DWORD G723BitRate;
} MEDIA_CAPS[MAX_MEDIA_CAPS_PER_DEVICE];
```

This data structure defines each codec that is supported on a line. The limit specifies 18. The following description applies for each member in the MEDIA\_CAPS data structure:

MediaPayload is an enumerated integer that contains one of the following values.

```
enum
{
    Media_Payload_G711Alaw64k = 2,
    Media_Payload_G711Alaw56k = 3, // "restricted"
    Media_Payload_G711Ulaw64k = 4,
    Media_Payload_G711Ulaw56k = 5, // "restricted"
    Media_Payload_G722_64k = 6,
    Media_Payload_G722_56k = 7,
    Media_Payload_G722_48k = 8,
    Media_Payload_G7231 = 9,
    Media_Payload_G728 = 10,
    Media_Payload_G729 = 11,
    Media_Payload_G729AnnexA = 12,
    Media_Payload_G729AnnexB = 15,
    Media_Payload_G729AnnexAwAnnexB = 16,
    Media_Payload_GSM_Full_Rate = 18,
    Media_Payload_GSM_Half_Rate = 19,
    Media_Payload_GSM_Enhanced_Full_Rate = 20,
    Media_Payload_Wide_Band_256k = 25,
    Media_Payload_Data64 = 32,
    Media_Payload_Data56 = 33,
    Media_Payload_GSM = 80,
    Media_Payload_G726_32K = 82,
    Media_Payload_G726_24K = 83,
    Media_Payload_G726_16K = 84,
    // Media_Payload_G729_B = 85,
    // Media_Payload_G729_B_LOW_COMPLEXITY = 86,
} Media_PayloadType;
```

MaxFramesPerPacket should read as MaxPacketSize and comprises a 16 bit integer that is specified in milliseconds. It indicates the RTP packet size. Typically, this value gets set to 20.

G723BitRate comprises a six byte field that contains either the G.723.1 information bit rate, or gets ignored. The values for the G.723.1 field comprises values that are enumerated as follows.

```
enum
{
Media_G723BRate_5_3 = 1, //5.3Kbps
Media_G723BRate_6_4 = 2 //6.4Kbps
} Media_G723BitRate;
```

## Setting RTP Parameters for Call

The `CCiscoLineDevSpecificSetRTPParamsForCall` class sets the RTP parameters for a specific call.



### Note

This extension only applies if extension version 0x00040000 or higher gets negotiated.

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificSetRTPParamsForCall
```

## Class Details

```
class CciscoLineDevSpecificSetRTPParamsForCall: public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificSetRTPParamsForCall () :
CCiscoLineDevSpecific(SLDST_USER_SET RTP_INFO) {}
    virtual ~ CciscoLineDevSpecificSetRTPParamsForCall () {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    // subtract out the virtual function table pointer
    DWORD m_RecieveIP; // UDP audio reception IP
    DWORD m_RecievePort; // UDP audio reception port
};
```

## Parameters

DWORD `m_MsgType`

Equals `SLDST_USER_SET RTP_INFO`

DWORD `m_ReceiveIP`

This specifies the RTP audio reception IP address in the network byte order to set for the call.

DWORD `m_ReceivePort`

This specifies the RTP audio reception port in the network byte order to set for the call.

## Redirect Set Original Called ID

The `CCiscoLineDevSpecificRedirectSetOrigCalled` class redirects a call to another party while it sets the original called ID of the call to any other party.



### Note

This extension only applies if extension version 0x00040000 or higher gets negotiated.

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificRedirectSetOrigCalled
```

## Class Details

```
class CCiscoLineDevSpecificRedirectSetOrigCalled: public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificRedirectSetOrigCalled () :
CCiscoLineDevSpecific(SLDST_REDIRECT_SET_ORIG_CALLED) {}
    virtual ~ CCiscoLineDevSpecificRedirectSetOrigCalled () {}
    char m_DestDirn[25];
    char m_SetOriginalCalledTo[25];
    // subtract virtual function table pointer
    virtual DWORD dwSize(void) const {return (sizeof (*this) - 4) ;
}
}
```

## Parameters

DWORD `m_MsgType`

Equals `SLDST_REDIRECT_SET_ORIG_CALLED`

char `m_DestDirn[25]`

Indicates the destination of the redirect. If this request is being used to transfer to voice mail, set this field to the voice mail pilot number of the DN of the line for the voice mail, to which you want to transfer.

char `m_SetOriginalCalledTo[25]`

Indicates the DN to which the `OriginalCalledParty` needs to be set. If this request is being used to transfer to voice mail, set this field to the DN of the line for the voice mail, to which you want to transfer.

HCALL `hCall` (in `lineDevSpecific` parameter list)

Equals the handle of the connected call.

## Join

The `CCiscoLineDevSpecificJoin` class joins two or more calls into one conference call. Each call that is being joined can be in the `ONHOLD` or the `CONNECTED` call state.

The Cisco Unified Communications Manager may succeed in joining some calls that are specified in the Join request, but not all. In this case, the Join request will succeed and the Cisco Unified Communications Manager attempts to join as many calls as possible.



### Note

This extension only applies if extension version 0x00040000 or higher gets negotiated.

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificJoin
```

## Class Details

```
class CCiscoLineDevSpecificJoin : public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificJoin () : CCiscoLineDevSpecific(SLDST_JOIN) {}
    virtual ~ CCiscoLineDevSpecificJoin () {}
    DWORD m_CallIDsToJoinCount;
    CALLIDS_TO_JOIN m_CallIDsToJoin;
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    // subtract out the virtual function table pointer
};
```

## Parameters

DWORD `m_MsgType`

Equals `SLDST_JOIN`

DWORD `m_CallIDsToJoinCount`

The number of callIDs that are contained in the `m_CallIDsToJoin` parameter.

CALLIDS\_TO\_JOIN `m_CallIDsToJoin`

A data structure that contains an array of `dwCallIDs` to join with the following format:

```
typedef struct {
    DWORD CallID; // dwCallID to Join
} CALLIDS_TO_JOIN[MAX_CALLIDS_TO_JOIN];
```

where `MAX_CALLIDS_TO_JOIN` is defined as:

```
const DWORD MAX_CALLIDS_TO_JOIN = 14;
```

HCALL `hCall` (in `LineDevSpecific` parameter list)

Equals the handle of the call that is being joined with `callIDsToJoin` to create the conference.

## Set User SRTP Algorithm IDs

The `CciscoLineDevSpecificUserSetSRTPAlgorithmID` class gets used to allow applications to set SRTP algorithm IDs. To use this class, ensure the `lineNegotiateExtVersion` API is called before opening the line. When calling `lineNegotiateExtVersion`, ensure the highest bit or second highest bit is set on both the `dwExtLowVersion` and `dwExtHighVersion` parameters. This causes the call to `lineOpen` to behave differently. The line does not actually open, but waits for a `lineDevSpecific` call to complete the open with more information. Provide the extra information that is required in the `CciscoLineDevSpecificUserSetSRTPAlgorithmID` class.



**Note** This extension is only available if extension version 0x80070000, 0x40070000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificUserSetSRTPAlgorithmID
```

### Procedure

- 
- Step 1** Call `lineNegotiateExtVersion` for the `deviceId` of the line that is to be opened. (0x80070000 or 0x40070000 with the `dwExtLowVersion` and `dwExtHighVersion` parameters)
  - Step 2** Call `lineOpen` for the `deviceId` of the line that is to be opened.
  - Step 3** Call `lineDevSpecific` with a `CciscoLineDevSpecificUserSetSRTPAlgorithmID` message in the `lpParams` parameter to specify SRTP algorithm IDs.
  - Step 4** Call `lineDevSpecific` with either `CciscoLineDevSpecificPortRegistrationPerCall` or `CCiscoLineDevSpecificUserControlRTPStream` message in the `lpParams` parameter.
- 

## Class Detail

```
class CciscoLineDevSpecificUserSetSRTPAlgorithmID: public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificUserSetSRTPAlgorithmID () :
        CCiscoLineDevSpecific(SLDST_USER_SET_SRTP_ALGORITHM_ID),
        m_SRTPAlgorithmCount(0),
        m_SRTP_Fixed_Element_Size(4)
    {
    }

    virtual ~ CciscoLineDevSpecificUserSetSRTPAlgorithmID () {}
    DWORD m_SRTPAlgorithmCount; //Maximum is MAX_CISCO_SRTP_ALGORITHM_IDS
    DWORD m_SRTP_Fixed_Element_Size; //Should be size of DWORD, it should be always 4.
    DWORD m_SRTPAlgorithm_Offset; //offset from beginning of the message buffer
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;} // subtract out the virtual
function table pointer
};
```

## Supported Algorithm Constants

```
enum CiscoSRTPAlgorithmIDs
{
    SRTP_NO_ENCRYPTION=0,
```

```

    SRTP_AES_128_COUNTER=1
};

```

## Parameters

DWORD m\_MsgType

Equals SLDST\_USER\_SET\_SRTP\_ALGORITHM\_ID

DWORD m\_SRTPAAlgorithmCount

This numbers of algorithm IDs that are specified in this message.

DWORD m\_SRTP\_Fixed\_Element\_Size

Should be size of DWORD, it should be always 4.

DWORD m\_SRTPAAlgorithm\_Offset

Offset from the beginning of the message buffer. This is offset where you start put algorithm ID array.



### Note

Be aware that the dwSize should be recalculated based on size of the structure, m\_SRTPAAlgorithmCount and m\_SRTP\_Fixed\_Element\_Size.

## Explicit Acquire

The CCiscoLineDevSpecificAcquire class gets used to explicitly acquire any CTI controllable device.

If a Superprovider application needs to open any CTI Controllable device on the Cisco Unified Communications Manager system, the application should explicitly acquire that device by using the above interface. After successful response, it can open the device as usual.



### Note

Be aware that this extension is only available if extension version 0x00070000 or higher is negotiated.

```

CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificAcquire

```

## Class Details

```

class CCiscoLineDevSpecificAcquire : public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificAcquire () : CCiscoLineDevSpecific(SLDST_ACQUIRE) {}
    virtual ~ CCiscoLineDevSpecificAcquire () {}
    char m_DeviceName[16];
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    // subtract out the virtual function table pointer
};

```

## Parameters

DWORD m\_MsgType

Equals SLDST\_ACQUIRE



```
m_DeviceName[16]
```

The DeviceName that needs to be explicitly acquired.

## Explicit De-Acquire

The CCiscoLineDevSpecificDeacquire class is used to explicitly de-acquire the explicitly acquired device.

If a Superprovider application has explicitly acquired any CTI Controllable device on the Cisco Unified Communications Manager system, then the application should explicitly De-acquire that device by using the above interface.



### Note

Be aware that this extension is only available if extension version 0x00070000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificDeacquire
```

## Class Details

```
class CCiscoLineDevSpecificDeacquire : public CCiscoLineDevSpecific
{
    public:
    CCiscoLineDevSpecificDeacquire () : CCiscoLineDevSpecific(SLDST_ACQUIRE) {}
    virtual ~ CCiscoLineDevSpecificDeacquire () {}
    char m_DeviceName[16];
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    // subtract out the virtual function table pointer
};
```

## Parameters

DWORD m\_MsgType

Equals SLDST\_DEACQUIRE

```
char m_DeviceName[16]
```

The DeviceName that needs to be explicitly de-acquired.

## Redirect FAC CMC

The `CCiscoLineDevSpecificRedirectFACCMC` class is used to redirect a call to another party that requires a FAC, CMC, or both.



### Note

Be aware that this extension is only available if extension version 0x00050000 or higher is negotiated.

```
CCiscoLineDevSpecific
```

```
|
```

```
+--CCiscoLineDevSpecificRedirectFACCMC
```

If the FAC is invalid, the TSP will return a new device-specific error code

`LINEERR_INVALIDFAC`. If the CMC is invalid, the TSP will return a new device-specific error code `LINEERR_INVALIDCMC`.

## Class Detail

```
class CCiscoLineDevSpecificRedirectFACCMC: public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificRedirectFACCMC () : CCiscoLineDevSpecific(SLDST_REDIRECT_FAC_CMC)
{}
    virtual ~ CCiscoLineDevSpecificRedirectFACCMC () {}
    char m_DestDirn[49];
    char m_FAC[17];
    char m_CMC[17];
    // subtract virtual function table pointer
    virtual DWORD dwSize(void) const {return (sizeof (*this) - 4) ;
}
}
```

## Parameters

DWORD `m_MsgType`

Equals `SLDST_REDIRECT_FAC_CMC`

char `m_DestDirn[49]`

Indicates the destination of the redirect.

char `m_FAC[17]`

Indicates the FAC digits. If the application does not want to pass any FAC digits, it must set this parameter to a NULL string.

char `m_CMC[17]`

Indicates the CMC digits. If the application does not want to pass any CMC digits, it must set this parameter to a NULL string.

HCALL `hCall` (in `lineDevSpecific` parameter list)

Equals the handle of the call to be redirected.

## Blind Transfer FAC CMC

The `CCiscoLineDevSpecificBlindTransferFACCMC` class is used to blind transfer a call to another party that requires a FAC, CMC, or both. If the FAC is invalid, the TSP will return a new device specific error code `LINEERR_INVALIDFAC`. If the CMC is invalid, the TSP will return a new device specific error code `LINEERR_INVALIDCMC`.



### Note

Be aware that this extension is only available if extension version 0x00050000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+--CCiscoLineDevSpecificBlindTransferFACCMC
```

## Class Detail

```
class CCiscoLineDevSpecificBlindTransferFACCMC: public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificBlindTransferFACCMC () :
CCiscoLineDevSpecific(SLDST_BLIND_TRANSFER_FAC_CMC) {}
    virtual ~ CCiscoLineDevSpecificBlindTransferFACCMC () {}
    char m_DestDirn[49];
    char m_FAC[17];
    char m_CMC[17];
    // subtract virtual function table pointer
    virtual DWORD dwSize(void) const {return (sizeof (*this) - 4) ;
}
}
```

## Parameters

`DWORD m_MsgType`

Equals `SLDST_BLIND_TRANSFER_FAC_CMC`

`char m_DestDirn[49]`

Indicates the destination of the blind transfer.

`char m_FAC[17]`

Indicates the FAC digits. If the application does not want to pass any FAC digits, it must set this parameter to a NULL string.

`char m_CMC[17]`

Indicates the CMC digits. If the application does not want to pass any CMC digits, it must set this parameter to a NULL string.

`HCALL hCall` (in `lineDevSpecific` parameter list)

Equals the handle of the call that is to be blind transferred.

## CTI Port Third Party Monitor

The `CCiscoLineDevSpecificCTIPortThirdPartyMonitor` class is used for opening CTI ports in third-party mode.

To use this class, ensure the `lineNegotiateExtVersion` API is called before opening the line. When calling `lineNegotiateExtVersion`, ensure the highest bit is set on both the `dwExtLowVersion` and `dwExtHighVersion` parameters. This causes the call to `lineOpen` to behave differently. The line does not actually open, but waits for a `lineDevSpecific` call to complete the open with more information. Provide the extra information that is required in the `CCiscoLineDevSpecificCTIPortThirdPartyMonitor` class.

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificCTIPortThirdPartyMonitor
```

### Procedure

- 
- Step 1** Call `lineNegotiateExtVersion` for the `deviceID` of the line that is to be opened. (OR `0x80000000` with the `dwExtLowVersion` and `dwExtHighVersion` parameters)
  - Step 2** Call `lineOpen` for the `deviceID` of the line that is to be opened.
  - Step 3** Call `lineDevSpecific` with a `CCiscoLineDevSpecificCTIPortThirdPartyMonitor` message in the `lpParams` parameter.
- 



#### Note

Be aware that this extension is only available if extension version `0x00050000` or higher is negotiated.

## Class Detail

```
class CCiscoLineDevSpecificCTIPortThirdPartyMonitor: public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificCTIPortThirdPartyMonitor () :
        CCiscoLineDevSpecific(SLDST_CTI_PORT_THIRD_PARTY_MONITOR) {}
    virtual ~ CCiscoLineDevSpecificCTIPortThirdPartyMonitor () {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;} //
        subtract out the virtual function table pointer
};
```

## Parameters

DWORD `m_MsgType`

equals `SLDST_CTI_PORT_THIRD_PARTY_MONITOR`

## Send Line Open

The `CciscoLineDevSpecificSendLineOpen` class is used for general delayed open purpose. To use this class, ensure the `lineNegotiateExtVersion` API is called before opening the line. When calling `lineNegotiateExtVersion`, ensure the second highest bit is set on both the `dwExtLowVersion` and `dwExtHighVersion` parameters. This causes the call to `lineOpen` to behave differently. The line does not actually open, but waits for a `lineDevSpecific` call to complete the open with more information. The extra information required is provided in the `CciscoLineDevSpecificUserSetSRTPAlgorithmID` class.

```
CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificSendLineOpen
```

### Procedure

- 
- Step 1** Call `lineNegotiateExtVersion` for the `deviceID` of the line that is to be opened. (0x40070000 with the `dwExtLowVersion` and `dwExtHighVersion` parameters).
  - Step 2** Call `lineOpen` for the `deviceID` of the line that is to be opened.
  - Step 3** Call other `lineDevSpecific`, like `CciscoLineDevSpecificUserSetSRTPAlgorithmID` message in the `lpParams` parameter to specify SRTP algorithm IDs.
  - Step 4** Call `lineDevSpecific` with either `CciscoLineDevSpecificSendLineOpen` to trigger the lineopen from TSP side.
- 



#### Note

Be aware that this extension is only available if extension version 0x40070000 or higher is negotiated.

---

## Class Detail

```
class CciscoLineDevSpecificSendLineOpen: public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificSendLineOpen () :
        CCiscoLineDevSpecific(SLDST_SEND_LINE_OPEN) {}

    virtual ~ CciscoLineDevSpecificSendLineOpen () {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;} // subtract out the virtual
function table pointer
};
```

## Set Intercom SpeedDial

Use the `CciscoLineSetIntercomSpeeddial` class to allow application to set or reset SpeedDial/Label on an intercom line.



### Note

Be aware that this extension is only available if extension version 0x00080000 or higher is negotiated

```
CCiscoLineDevSpecific
|
+-- CciscoLineSetIntercomSpeeddial
```

### Procedure

- 
- Step 1** Call `lineNegotiateExtVersion` for the deviceID of the line that is to be opened (0x00080000 or higher).
  - Step 2** Call `lineOpen` for the deviceID of the line that is to be opened.
  - Step 3** Wait for line in service.
  - Step 4** Call `CciscoLineSetIntercomSpeeddial` to set or reset speed dial setting on the intercom line.
- 

## Class Detail

```
class CciscoLineSetIntercomSpeeddial: public CCiscoLineDevSpecific
{
public:
    CciscoLineSetIntercomSpeeddial () :
        CCiscoLineDevSpecific(SLDST_LINE_SET_INTERCOM_SPEEDDIAL) {}

    virtual ~ CciscoLineSetIntercomSpeeddial () {}
    DWORD SetOption;          //0=clear app value, 1= set App Value
    char Intercom_DN[MAX_DIRN];
    char Intercom_Ascii_Label[MAX_DIRN];
    wchar_t Intercom_Unicode_Label[MAX_DIRN];
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;} // subtract out the virtual
function table pointer
};
```

## Parameters

DWORD `m_MsgType`

Equals `SLDST_USER_SET_INTERCOM_SPEEDDIAL`

DWORD `SetOption`

Use this parameter to indicate whether the application wants to set a new intercom speed dial value or clear the previous value. 0 = clear, 1 = set.

Char `Intercom_DN [MAX_DIRN]`

A DN array that indicates the intercom target

Char `Intercom_Ascii_Label[MAX_DIRN]`

Indicates the ASCII value of the intercom line label

Wchar\_tIntercom\_Unicode\_Label[MAX\_DIRN]

Indicates the Unicode value of the intercom line label

MAX\_DIRN is defined as 25.

## Intercom Talk Back

Use the CciscoLineIntercomTalkback class to allow the application to initiate talk back on an incoming intercom call on an intercom line.



### Note

Be aware that this extension is only available if extension version 0x00080000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+-- CciscoLineIntercomTalkback
```

## Class Detail

```
class CciscoLineIntercomTalkback: public CCiscoLineDevSpecific
{
public:
    CciscoLineIntercomTalkback () :
        CCiscoLineDevSpecific(SLDST_INTERCOM_TALKBACK) {}

    virtual ~ CciscoLineIntercomTalkback () {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;} // subtract out the virtual
function table pointer
};
```

## Redirect with Feature Priority

CciscoLineRedirectWithFeaturePriority enables an application to redirect calls with specified feature priorities. The following is the structure of CciscoLineDevSpecific:

```
CCiscoLineDevSpecific
|
+-- CciscoLineRedirectWithFeaturePriority
```



### Note

Be aware that this extension is only available if the extension version 0x00080001 or higher is negotiated.

## Detail

```
class CciscoLineRedirectWithFeaturePriority: public CCiscoLineDevSpecific
{
public:
    CciscoLineRedirectWithFeaturePriority() :
        CCiscoLineDevSpecific(SLDST_REDIRECT_WITH_FEATURE_PRIORITY) {}

    virtual ~ CciscoLineRedirectWithFeaturePriority () {}
    CiscoDoNotDisturbFeaturePriority FeaturePriority;
    char m_DestDirn[25];
```

```
virtual DWORD dwSize(void) const {return sizeof(*this)-4;} // subtract out the virtual
function table pointer
};
```

## Parameters

DWORD m\_MsgType

Equals SLDST\_REDIRECT\_WITH\_FEATURE\_PRIORITY

enum CiscoDoNotDisturbFeaturePriority {CallPriority\_NORMAL = 1, CallPriority\_URGENT = 2, CallPriority\_EMERGENCY = 3};

This identifies the priorities.

char m\_DestDirn[25];

This is redirect destination.

## Start Call Monitoring

Use `CCiscoLineDevSpecificStartCallMonitoring` to allow application to send a start monitoring request for the active call on a line.



### Note

Be aware that this extension is only available if extension version 0x00080000 or higher is negotiated.

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificStartCallMonitoring
```

## Class Detail

```
class CCiscoLineDevSpecificStartCallMonitoring: public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificStartCallMonitoring () :
CCiscoLineDevSpecific(SLDST_START_CALL_MONITORING) {}
    virtual ~    CCiscoLineDevSpecificStartCallMonitoring () {}
    DWORD m_PermanentLineID ;
    DWORD m_MonitorMode;
    DWORD m_ToneDirection;
    // subtract out the virtual function table pointer
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
} ;
```

## Parameters

DWORD m\_MsgType

Equals SLDST\_START\_MONITORING

DWORD m\_PermanentLineID

The permanent lineID of the line whose active call has to be monitored.

DWORD MonitorMode

This can have the following enum value:



```
enum
{
    MonitorMode_None = 0,
    MonitorMode_Silent = 1,
    MonitorMode_Whisper = 2,    // Not used
    MonitorMode_Active = 3     // Not used
} MonitorMode;
```

**Note**

Silent Monitoring mode represents the only mode that is supported in which the supervisor cannot talk to the agent.

**DWORD PlayToneDirection**

This parameter specifies whether a tone should play at the agent or customer phone when monitoring starts. It can have following enum values:

```
enum
{
    PlayToneDirection_LocalOnly = 0,
    PlayToneDirection_RemoteOnly,
    PlayToneDirection_BothLocalAndRemote,
    PlayToneDirection_NoLocalOrRemote
} PlayToneDirection
```

**Return Values**

- LINERR\_OPERATIONFAILED
- LINEERR\_OPERATIONUNAVAIL
- LINEERR\_RESOURCEUNAVAIL
- LINEERR\_BIB\_RESOURCE\_UNAVAIL
- LINERR\_PENDING\_REQUEST
- LINEERR\_OPERATION\_ALREADY\_INPROGRESS
- LINEERR\_ALREADY\_IN\_REQUESTED\_STATE
- LINEERR\_PRIMARY\_CALL\_INVALID
- LINEERR\_PRIMARY\_CALL\_STATE\_INVALID

**Start Call Recording**

Use `CCiscoLineDevSpecificStartCallRecording` to allow applications to send a recording request for the active call on that line.

**Note**

Be aware that this extension is only available if extension version 0x00080000 or higher is negotiated

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificStartCallRecording
```

**Class Detail**

```
class CCiscoLineDevSpecificStartCallRecording: public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificStartCallRecording () :
    CCiscoLineDevSpecific(SLDST_START_CALL_RECORDING) {}
    virtual ~ CCiscoLineDevSpecificStartCallRecording () {}
```

```

    DWORD m_ToneDirection;
    // subtract out the virtual function table pointer
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
} ;

```

## Parameters

DWORD m\_MsgType

Equals SLDST\_START\_RECORDING

DWORD PlayToneDirection

This parameter specifies whether a tone should play at the agent or customer phone when recording starts. It can have following enum values:

```

enum
{
    PlayToneDirection_NoLocalOrRemote = 0,
    PlayToneDirection_LocalOnly,
    PlayToneDirection_RemoteOnly,
    PlayToneDirection_BothLocalAndRemote
} PlayToneDirection

```

## Return Values

- LINERR\_OPERATIONFAILED
- LINEERR\_OPERATIONUNAVAIL
- LINEERR\_INVALIDCALLHANDLE
- LINEERR\_BIB\_RESOURCE\_UNAVAIL
- LINERR\_PENDING\_REQUEST
- LINERR\_OPERATION\_ALREADY\_INPROGRESS

## StopCall Recording

Use `CCiscoLineDevSpecificStopCallRecording` to allow application to stop recording a call on that line.



### Note

Be aware that this extension is only available if extension version 0x00080000 or higher is negotiated.

```

CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificStopCallRecording

```

## Class Detail

```

class CCiscoLineDevSpecificStopCallRecording: public CCiscoLineDevSpecific
{
public:
    CCiscoLineDevSpecificStopCallRecording () :
    CCiscoLineDevSpecific(SLDST_STOP_CALL_RECORDING) {}
    virtual ~    CCiscoLineDevSpecificStopCallRecording () {}

    // subtract out the virtual function table pointer
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
} ;

```

## Parameters

DWORD m\_MsgType  
 Equals SLDST\_STOP\_RECORDING

## Return Values

- LINERR\_OPERATIONFAILED
- LINEERR\_OPERATIONUNAVAIL
- LINEERR\_INVALCALLHANDLE
- LINERR\_PENDING\_REQUEST

## Set IPv6 Address and Mode

Use the `CciscoLineDevSpecificSetIPv6AddressAndMode` class to allow the application to set IPv6 address and addressing mode during the static registration. To use this class, ensure the `lineNegotiateExtVersion` API must be called before opening the line. When calling `lineNegotiateExtVersion`, ensure the highest bit or the second highest must be set on both the `dwExtLowVersion` and `dwExtHighVersion` parameters. This causes the call to `lineOpen` to behave differently. The line does not actually open, but waits for a `lineDevSpecific` call to complete the open with more information. The extra information required is provided in the `CciscoLineDevSpecificSetIPv6AddressAndMode` class.

```
CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificSetIPv6AddressAndMode
```



### Note

This extension is available only if extension version 0x80090000, 0x40090000 or higher is negotiated.

### Procedure

- 
- Step 1** Open Call `lineNegotiateExtVersion` for the deviceID of the line (0x80090000 or 0x40090000 with the `dwExtLowVersion` and `dwExtHighVersion` parameters)
  - Step 2** Open Call `lineOpen` for the deviceID of the line.
  - Step 3** Call `lineDevSpecific` with a `CciscoLineDevSpecificSetIPv6AddressAndMode` message in the `lpParams` parameter to specify IPv6 address and the IP Addressing mode as IPv6.
- 

## Class Detail

```
class CciscoLineDevSpecificSetIPv6AddressAndMode: public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificSetIPv6AddressAndMode() :
        CCiscoLineDevSpecific(SLDST_USER_SET_IPV6_ADDRESS_AND_MODE), m_ReceivePort(-1),
        m_IPAddressMode(
            (IpAddressingMode) 1)
    {
    }
    virtual ~ CciscoLineDevSpecificSetIPv6AddressAndMode()
    {
    }
```

```

    }
    char m_ReceiveIPv6Address[16];
    DWORD m_ReceivePort;
    IpAddressingMode m_IPAddressMode;
    virtual DWORD dwSize(void) const
    {
        return sizeof(*this) - 4;
    } // subtract out the virtual function table pointer
};

```

## Parameters

DWORD m\_MsgType

Equals SLDST\_USER\_SET\_IPv6\_ADDRESS

Charm\_ReceiveIPv6Address[16]

User has to specify the IPv6 address to register the CTI Port with

DWORD m\_ReceivePort

This specifies the port number for the user to register the CTI Port.

Intm\_IPAddressMode

This specifies the Addressing mode with which user wants the CTI Port/RP registered.

## Set RTP Parameters for IPv6 Calls

Use CciscoLineDevSpecificSetRTPParamsForCallIPv6 class to set the RTP parameters for calls for which you must specify IPv6 address.



### Note

Be aware that this extension is available only if extension version 0x00090000 or higher is negotiated.

## Class Detail

```

class CciscoLineDevSpecificSetRTPParamsForCallIPv6: public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificSetRTPParamsForCallIPv6 () :
    CCiscoLineDevSpecific(SLDST_USER_SET RTP_INFO_IPv6) {}
    virtual ~ CciscoLineDevSpecificSetRTPParamsForCallIPv6 () {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;} // subtract out the virtual
    function table pointer
    char m_RecieveIPv6[16]; // UDP audio reception IPv6
    DWORD m_RecievePort // UDP audio reception port
};

```

## Parameters

DWORD m\_MsgType

Equals SLDST\_USER\_SET RTP\_INFO\_IPv6

DWORD m\_ReceiveIPv6

This is the RTP audio reception IPv6 address to set for the call

DWORD m\_RecievePort

This is the RTP audio reception port to set for the call.

## Direct Transfer

Use the CciscoLineDevSpecificDirectTransfer to transfer calls across lines or on the same line.

```
CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificDirectTransfer
```



### Note

Be aware that this extension is available only if extension version 0x00090001 or higher is negotiated.

## Class Detail

```
class CciscoLineDevSpecificDirectTransfer: public CCiscoLineDevSpecific
{
    public:
        CciscoLineDevSpecificDirectTransfer () :
CCiscoLineDevSpecific(SLDST_DIRECT_TRANSFER) {}
        virtual ~ CciscoLineDevSpecificDirectTransfer () {}
        DWORD m_CallIDsToTransfer;
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

## Parameters

DWORD m\_MsgType

equals SLDST\_DIRECT\_TRANSFER

DWORD m\_CallIDsToTransfer

Consult dwCallID to be transferred

HCALL hCall (in LineDevSpecific parameter list)

Equals the handle of the call that is being transferred.

## RegisterCallPickUpGroupForNotification

The `CciscoLineDevSpecificRegisterCallPickupGroupForNotification` class is used to register the call Pickup Group for notification on calls for Pickup.

```
CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificRegisterCallPickupGroupForNotification
```



### Note

This extension is available only if extension version 0x000A0000 or higher is negotiated.

## Class Detail

```
class CciscoLineDevSpecificRegisterCallPickupGroupForNotification: public
CCiscoLineDevSpecific
{
    public:
        CciscoLineDevSpecificRegisterCallPickupGroupForNotification ():
CCiscoLineDevSpecific (SLDST_CALLPICKUP_GROUP_REGISTER) {}
        virtual ~ CciscoLineDevSpecificRegisterCallPickupGroupForNotification () {}
        char callPickupGroupDN[MAX_DIRN];
        char callPickupGroupPartition[MAX_PARTITION];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

## Parameters

DWORD `m_MsgType`

equals `SLDST_CALLPICKUP_GROUP_REGISTER`

Char `CallPickupGroupDN []` - DN of the pickup Group

Char `CallPickupGroupPartition []` - Partition of the PickupGroup

## UnRegisterCallPickUpGroupForNotification

The `CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification` class is used to unregister the call Pickup Group for notification on calls for Pickup.

```
CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification
```



### Note

This extension is available only if extension version 0x000A0000 or higher is negotiated

## Class Details

```
class CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification: public
CCiscoLineDevSpecific
{
    Public:
        CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification () :
CCiscoLineDevSpecific(SLDST_CALLPICKUP_GROUP_UNREGISTER) {}
```

```

        virtual ~ CciscoLineDevSpecificUnRegisterCallPickupGroupForNotification () {}
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};

```

## Parameters

DWORD m\_MsgType  
 equals SLDST\_CALLPICKUP\_GROUP\_UNREGISTER

## CallPickUpRequest

The CciscoLineDevSpecificCallPickupRequest class is used to Pickup the call from the PickGroup.

```

CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificCallPickupRequest

```



### Note

This extension is available only if extension version 0x000A0000 or higher is negotiated.

## Class Details

```

class CciscoLineDevSpecificCallPickupRequest: public CCiscoLineDevSpecific
{
    public:
        CciscoLineDevSpecificCallPickupRequest (): CCiscoLineDevSpecific
        (SLDST_CALLPICKUP_CALL) {}
        virtual ~ CciscoLineDevSpecificCallPickupRequest () {}
        DWORD PickupType;
        char PickupGroupDN[MAX_DIRN];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};

```

## Parameters

DWORD m\_MsgType  
 equals SLDST\_CALLPICKUP\_CALL  
 Char PickupGroupDN [] - DN of the pickup Group/DN;will be required for GroupCallPickUp and DirectedCallPickUp  
 DWORD PickupType - indicates the type of pickup (CtiCallPickUp, CtiGroupCallPickUp, , CtiOtherPickup, DirectedCallPickup)

```

enum CallPickupType{
    CallPickup_Simple = 0,
    CallPickup_Group = 1,
    CallPickup_Other = 2,
    CallPickup_Direct = 3
};

```

## Start Send Media To BIB

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificStartSendMediaToBIBRequest
```

### Description

The `CCiscoLineDevSpecificStartSendMediaToBIBRequest` class allows the application to initiate agent greeting to the customer call.



#### Note

---

This extension is only available if extension version 0x000B0000 or higher is negotiated.

---

TAPI line handle and TAPI call handle are required for this request.

### Class Detail

```
class CCiscoLineDevSpecificStartSendMediaToBIBRequest: public CCiscoLineDevSpecific
{
    public:
        CCiscoLineDevSpecificStartSendMediaToBIBRequest (): CCiscoLineDevSpecific
(SLDST_START_SEND_MEDIA_TO_BIB) {}
        virtual ~ CCiscoLineDevSpecificStartSendMediaToBIBRequest () {}
        char m_IVRDN [49];
        char m_CGPNTTOIVR [49];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
```

### Parameters

DWORD `m_MsgType`

equals `SLDST_START_SEND_MEDIA_TO_BIB`

char `m_IVRDN` [49]

IVR port DN where Agent Greeting will be played from

char `m_CGPNTTOIVR` [49]

The `CallingPartyDN` passed to IVR. The application can use this field to pass DN as `CallingPartyDN` for the agent greeting call.



## Stop Send Media To BIB

```
CCiscoLineDevSpecific
|
+-- CCiscoLineDevSpecificStopSendMediaToBIBRequest
```

### Description

The `CCiscoLineDevSpecificStopSendMediaToBIBRequest` class allows the application to stop the agent greeting that is playing on the agent-to-customer call.



#### Note

---

This extension is only available if extension version 0x000B0000 or higher is negotiated.

---

TAPI line handle and TAPI call handle are required for this request.

### Class Detail

```
class CCiscoLineDevSpecificStopSendMediaToBIBRequest: public CCiscoLineDevSpecific
{
    public:
        CCiscoLineDevSpecificStopSendMediaToBIBRequest (): CCiscoLineDevSpecific
(SLDST_STOP_SEND_MEDIA_TO_BIB) {}
        virtual ~ CCiscoLineDevSpecificStopSendMediaToBIBRequest () {}
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

### Parameters

DWORD m\_MsgType  
 equals SLDST\_START\_SEND\_MEDIA\_TO\_BIB

## Agent Zip Tone

```
CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificEnableFeatureSupport
```

### Description

The `CCiscoLineDevSpecificPlaytone` class is used to play the tone (Zip Tone) to the direction specified in the request.



#### Note

---

This extension is only available if extension version 0x000B0000 or higher is negotiated.

---

### Class Detail

```
class CCiscoLineDevSpecificPlaytone: public CCiscoLineDevSpecific //AgentZiptone
{
```

```

public:
    CCiscoLineDevSpecificPlaytone() :
    CCiscoLineDevSpecific(SLDST_PLAY_TONE)
    {
    }
    virtual ~ CCiscoLineDevSpecificPlaytone()
    {
    }
    DWORD m_Tone;
    DWORD m_ToneDirection;
    virtual DWORD dwSize(void) const
    {
        return sizeof(*this) - 4;
    } // subtract out the virtual function table pointer
};

```

## Parameters

DWORD m\_Tone - Indicates the Tone type  
 equals CTONE\_ZIP

DWORD m\_ToneDirection - Indicates the direction of the tone to be played;  
 equals Tonedirection (Local/Remote)

## Early Offer

New error Code – LINEERR\_REGISTER\_GETPORT\_SUPPORT\_MISMATCH 0xC000000F

## Enable Feature

```

CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificEnableFeatureSupport

```

## Description

The CciscoLineDevSpecificEnableFeatureSupport class allows application to enhance or update feature support.



### Note

---

This extension is only available if extension version 0x000B0000 or higher is negotiated.

---

## Class Detail

```

class CciscoLineDevSpecificEnableFeatureSupport: public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificEnableFeatureSupport() :
        CCiscoLineDevSpecific(SLDST_ENABLE_FEATURE_SUPPORT)
    {
    }

    virtual ~ CciscoLineDevSpecificEnableFeatureSupport()

```

```

    {
    }
    DWORD m_Feature;
    DWORD m_Feature_Capability; //Should have Value supported for Feature specified in
m_Feature
    virtual DWORD dwSize(void) const
    {
        return sizeof(*this) - 4;
    } // subtract out the virtual function table pointer

```

## Parameters

DWORD m\_MsgType

equals SLDST\_ENABLE\_FEATURE\_SUPPORT

DWORD m\_Feature

Feature value for which the capability needs to be changed and should have a value from the following Enum:

```

enum TspFeatureSupport
{
    Feature_unknown = 0,
    Feature_EarlyOffer = 1
};
    DWORD m_Feature_Capability

```

The Capability information that needs to be changed/updated for the feature. This information changes depending on the feature.

Early Offer (Get Port) Support:

m\_Feature should be Feature\_EarlyOffer(1) and

m\_Feature\_Capability should have a value from following Enum:

```

enum TspFeatureOption
{
    Feature_Disable = 0,
    Feature_Enable = 1
};

```

Sample Code:

Here is a sample code that illustrates how applications must use this devspecific type, and fill the Class Object to enable/disable the Early Offer feature support.

```

void main()
{
    .....
    CiscoLineDevSpecificEnableFeatureSupport featureObject;
    featureObject.m_MsgType = SLDST_ENABLE_FEATURE_SUPPORT;
    featureObject.m_Feature = Feature_EarlyOffer(1);
    featureObject.m_Feature_Capability = Feature_Enable(1)/ Feature_Disable(0);

    int result = TSPI_lineDevSpecific(dwRequestID,hdLine, dwAddressID, NULL,
&featureObject, sizeof(CciscoLineDevSpecificEnableFeatureSupport));
    .....
    .....
}
New CiscoLineDevStateOutOfServiceReason_EMLogin and
CiscoLineDevStateOutOfServiceReason_EMLogout values in the CiscoLineDevStateOtherReason
enumeration type in CiscoLineDevSpecificMsg.h:
enum CiscoLineDevStateOutOfServiceReason
{

```

```

CiscoLineDevStateOutOfServiceReason_Unknown = 0x00000000,
CiscoLineDevStateOutOfServiceReason_CallManagerFailure = 0x00000001,
CiscoLineDevStateOutOfServiceReason_ReHomeToHigherPriorityCM =0x00000002,
CiscoLineDevStateOutOfServiceReason_NoCallManagerAvailable =0x00000003,
CiscoLineDevStateOutOfServiceReason_DeviceFailure =0x00000004,
CiscoLineDevStateOutOfServiceReason_DeviceUnregistered =0x00000005,
CiscoLineDevStateOutOfServiceReason_EnergyWisePowerSavePlus =0x00000006,
CiscoLineDevStateOutOfServiceReason_EMLogin = 0x00000007,
CiscoLineDevStateOutOfServiceReason_EMLogout =      0x00000008,
CiscoLineDevStateOutOfServiceReason_CtiLinkFailure =0x00000101
};
New CiscoLineDevStateCloseReason enumeration type in CiscoLineDevSpecificMsg.h:
enum CiscoLineDevStateCloseReason
{
    CiscoLineDevStateCloseReason_Unknown = 0,
    CiscoLineDevStateCloseReason_EMLogin = 1,
    CiscoLineDevStateCloseReason_EMLogout =2
};

New CiscoLineDevStateOtherReason enumeration type in CiscoLineDevSpecificMsg.h:
enum CiscoLineDevStateOtherReason
{
    CiscoLineDevStateOtherReason_Unknown = 0,
    CiscoLineDevStateOtherReason_OtherLineInactive =1,
    CiscoLineDevStateOtherReason_OtherLineActive = 2,
    CiscoLineDevStateOtherReason_OtherLineCapsChange =3
};

New LINEERR_DEVICE_INACTIVE error is returned if an operation is invoked on a line device
in "inactive" state.

```

## UpdateMonitorMode

```

CCiscoLineDevSpecific
|
+-- CciscoLineDevSpecificMonitoringUpdateMode

```

### Description

The CciscoLineDevSpecificMonitoringUpdateMode class allows the application to toggle between the silent monitoring and whisper coaching modes, and vice versa.



#### Note

---

This extension is only available if extension version 0x000B0000 or higher is negotiated.

---

### Class Detail

```

class CciscoLineDevSpecificMonitoringUpdateMode : public CCiscoLineDevSpecific
{
public:
    CciscoLineDevSpecificMonitoringUpdateMode (): CCiscoLineDevSpecific
(SLDST_UPDATE_MONITOR_MODE) {}
    virtual ~ CciscoLineDevSpecificMonitoringUpdateMode () {}
    DWORD m_MonitorMode;
    DWORD m_ActiveToneDirection;
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    // subtract out the virtual function table pointer
};

```

## Parameters

DWORD m\_MsgType  
 equals SLDST\_UPDATE\_MONITOR\_MODE

DWORD m\_MonitorMode  
 Monitoring mode to toggle to

DWORD m\_ActiveToneDirection  
 Direction of the tone to be played

## Cisco Line Device Feature Extensions

CCiscoLineDevSpecificFeature represents the parent class. Currently, it consist of only one subclass: CCiscoLineDevSpecificFeature\_DoNotDisturb, which allows applications to enable and disable the Do-Not-Disturb feature on a device.

This following sections describe the line device feature-specific extensions to the TAPI structures that Cisco TSP supports:

- [CCiscoLineDevSpecificFeature, page 6-61](#)
- [Do-Not-Disturb, page 6-62](#)
- [Do-Not-Disturb Change Notification Event, page 6-63](#)

## CCiscoLineDevSpecificFeature

This section provides information on how to invoke Cisco-specific TAPI extensions with the CCiscoLineDevSpecificFeature class, which represents the parent class to all the following classes.



### Note

---

Be aware that this virtual class is provided for informational purposes only.

---

CCiscoLineDevSpecificFeature

## Header File

The file CiscoLineDevSpecific.h contains the corresponding constant, structure, and class definitions for the Cisco lineDevSpecificFeature extension classes.

## Class Detail

```
class CCiscoLineDevSpecificFeature
{
public:
    CCiscoLineDevSpecificFeature(const DWORD msgType): m_MsgType(msgType) {}
    virtual ~ CCiscoLineDevSpecificFeature() {}
    DWORD GetMsgType(void) const {return m_MsgType;}
    void* lpParams(void) const {return (void*)&m_MsgType;}
    virtual DWORD dwSize(void) const = 0;
private:
    DWORD m_MsgType;
```

```
};
```

## Functions

lpParms()

Function that can be used to obtain a pointer to the parameter block

dwSize()

Function that returns size of the parameter block area

## Parameter

m\_MsgType

Specifies the type of message. The parameter value uniquely identifies the feature to invoke on the device. The PHONEBUTTONFUNCTION\_ TAPI\_Constants definition lists the valid feature identifiers. Currently, the only recognized value specifies PHONEBUTTONFUNCTION\_DONOTDISTURB (0x0000001A).

Each subclass of CCiscoLineDevSpecificFeature includes a unique value that is assigned to the m\_MsgType parameter.

## Subclasses

Each subclass of CCiscoLineDevSpecificFeature carries a unique value that is assigned to the m\_MsgType parameter. If you are using C instead of C++, this represents the first parameter in the structure.

## Do-Not-Disturb

Use the CCiscoLineDevSpecificFeature\_DoNotDisturb class in conjunction with the request to enable or disable the DND feature on a device.

The Do-Not-Disturb feature gives phone users the ability to go into a Do Not Disturb (DND) state on the phone when they are away from their phones or simply do not want to answer the incoming calls. A phone softkey, DND, allows users to enable or disable this feature.

```
CCiscoLineDevSpecificFeature
|
+-- CCiscoLineDevSpecificFeature_DoNotDisturb
```

## Class Detail

```
class CCiscoLineDevSpecificFeature_DoNotDisturb : public CCiscoLineDevSpecificFeature
{
public:
    CCiscoLineDevSpecificFeature_DoNotDisturb()
    : CCiscoLineDevSpecificFeature(PHONEBUTTONFUNCTION_DONOTDISTURB),
      m_Operation((CiscoDoNotDisturbOperation)0) {}
    virtual ~CCiscoLineDevSpecificFeature_DoNotDisturb() {}
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}

    CiscoDoNotDisturbOperation m_Operation;
};
```

## Parameters

DWORD m\_MsgType

Equals PHONEBUTTONFUNCTION\_DONOTDISTURB.

CiscoDoNotDisturbOperation m\_Operation

Specifies a requested operation and can comprise one of the following enum values:

```
enum CiscoDoNotDisturbOperation {
    DoNotDisturbOperation_ENABLE    = 1,
    DoNotDisturbOperation_DISABLE   = 2
};
```

## Do-Not-Disturb Change Notification Event

Cisco TSP notifies applications via the LINE\_DEVSPECIFICFEATURE message about changes in the DND configuration or status. To receive change notifications, an application needs to enable the DEVSPECIFIC\_DONOTDISTURB\_CHANGED message flag with a lineDevSpecific SLDST\_SET\_STATUS\_MESSAGES request.

The LINE\_DEVSPECIFICFEATURE message notifies the application about device-specific events that occur on a line device. In the case of a Do-Not-Disturb Change Notification, the message includes information about the type of change that occurred on a device and the resulting feature status or configured option.

## Message Details

```
LINE_DEVSPECIFICFEATURE
dwDevice = (DWORD) hLine;
dwCallbackInstance = (DWORD) hCallback;
dwParam1 = (DWORD) PHONEBUTTONFUNCTION_DONOTDISTURB;
dwParam2 = (DWORD) typeOfChange;
dwParam3 = (DWORD) currentValue;

enum CiscoDoNotDisturbOption {
    DoNotDisturbOption_NONE        = 0,
    DoNotDisturbOption_RINGEROFF   = 1,
    DoNotDisturbOption_REJECT      = 2
};

enum CiscoDoNotDisturbStatus {
    DoNotDisturbStatus_UNKNOWN     = 0,
    DoNotDisturbStatus_ENABLED     = 1,
    DoNotDisturbStatus_DISABLED    = 2
};

enum CiscoDoNotDisturbNotification {
    DoNotDisturb_STATUS_CHANGED    = 1,
    DoNotDisturb_OPTION_CHANGED    = 2
};
```

## Parameters

dwDevice

A handle to a line device

**dwCallbackInstance**

The callback instance that is supplied when the line is opened

**dwParam1**

Always equal to PHONEBUTTONFUNCTION\_DONOTDISTURB for the Do-Not-Disturb change notification

**dwParam2**

Indicates type of change and can comprise one of the following enum values:

```
enum CiscoDoNotDisturbNotification {
    DoNotDisturb_STATUS_CHANGED = 1,
    DoNotDisturb_OPTION_CHANGED = 2
};
```

**dwParam3**

If the dwParm2 indicates status change with the value DoNotDisturb\_STATUS\_CHANGED, this parameter can comprise one of the following enum values:

```
enum CiscoDoNotDisturbStatus {
    DoNotDisturbStatus_UNKNOWN = 0,
    DoNotDisturbStatus_ENABLED = 1,
    DoNotDisturbStatus_DISABLED = 2
};
```

If the dwParm2 indicates option change with the value DoNotDisturb\_OPTION\_CHANGED, this parameter can comprise one of the following enum values:

```
enum CiscoDoNotDisturbOption {
    DoNotDisturbOption_NONE = 0,
    DoNotDisturbOption_RINGEROFF = 1,
    DoNotDisturbOption_REJECT = 2
};
```

## Cisco Phone Device-Specific Extensions

Table 6-2 lists the subclasses of CiscoPhoneDevSpecific.

**Table 6-2** Cisco Phone Device-Specific TAPI Functions

Cisco Functions	Synopsis
<a href="#">CCiscoPhoneDevSpecific</a>	The CCiscoPhoneDevSpecific class represents the parent class to the following classes.
<a href="#">Device Data PassThrough</a>	Allows the application to send the Device Specific XSI data through CTI.
<a href="#">Explicit Acquire</a>	Allows the application to acquire any CTI-controllable device that can get opened later in superprovider mode.
<a href="#">Explicit De-Acquire</a>	Allows the application to deacquire a CTI-controllable device that was explicitly acquired.
<a href="#">Request Call RTP Snapshot</a>	Allows the application to request secure RTP indicator for calls on the device.



**Table 6-2** Cisco Phone Device-Specific TAPI Functions (continued)

Cisco Functions	Synopsis
<a href="#">Set Status Msgs</a>	Allows the application to set status bit map to enable specific DEVICE_DEVSPECIFIC messages to be sent to the application.
<a href="#">Set Unicode Display</a>	Sets the Unicode display on the phone.

## CCiscoPhoneDevSpecific

This section provides information on how to perform Cisco TAPI-specific functions with the CCiscoPhoneDevSpecific class, which represents the parent class to all the following classes.



### Note

Be aware that this virtual class is provided for informational purposes only.

```

CCiscoPhoneDevSpecific
|
+-- CCiscoPhoneDevSpecificDataPassThrough
|
+-- CCiscoPhoneDevSpecificSetStatusMsgs
|
+-- CCiscoPhoneDevSpecificSetUnicodeDisplay
|
+-- CCiscoPhoneDevSpecificAcquire
|
+-- CCiscoPhoneDevSpecificDeacquire
|
+-- CCiscoPhoneDevSpecificGetRTPSnapshot

```

## Header File

The file CiscoLineDevSpecific.h contains the constant, structure, and class definition for the Cisco phone device-specific classes.

## Class Detail

```

class CCiscoPhoneDevSpecific
{
    public :
        CCiscoPhoneDevSpecific(DWORD msgType) :m_MsgType(msgType) {}
        virtual ~CCiscoPhoneDevSpecific() {}
        DWORD GetMsgType (void) const { return m_MsgType;}
        void *lpParams(void) const {return (void*)&m_MsgType;}
        virtual DWORD dwSize(void) const = 0;
    private :
        DWORD m_MsgType ;
}

```

## Functions

lpParms()

Function that can be used to obtain the pointer to the parameter block  
 dwSize()

Function that will give the size of the parameter block area

## Parameter

m\_MsgType

Specifies the type of message.

## Subclasses

Each subclass of CCiscoPhoneDevSpecific represents a different value that is assigned to the parameter m\_MsgType. If you are using C instead of C++, this represents the first parameter in the structure.

## Enumeration

The CiscoPhoneDevSpecificType enumeration includes valid message identifiers.

```
enum CiscoPhoneDevSpecificType
{
    CPDST_DEVICE_DATA_PASSTHROUGH_REQUEST = 1,
    CPDST_SET_DEVICE_STATUS_MESSAGES,
    CPDST_SET_DEVICE_UNICODE_DISPLAY,
    CPDST_ACQUIRE,
    CPDST_DE_ACQUIRE,
    CPDST_REQUEST_DEVICE_SNAPSHOT_INFO
};
```

## Device Data Passthrough

XSI-enabled IP phones allow applications to directly communicate with the phone and access XSI features (for example, manipulate display, get user input, play tone, and so on). To allow TAPI applications to have access to some of these XSI capabilities without having to setup and maintain an independent connection directly to the phone, TAPI will provide the ability to send device data through the CTI interface. This feature gets exposed as a Cisco Unified TSP device-specific extension.

PhoneDevSpecificDataPassthrough request only gets supported for the IP phone devices. Application must open a TAPI phone device with minimum extension version 0x00030000 to make use of this feature.

The CCiscoPhoneDevSpecificDataPassThrough class is used to send the device-specific data to CTI-controlled IP phone devices.



### Note

Be aware that this extension requires applications to negotiate extension version as 0x00030000.

```
CCiscoPhoneDevSpecific
|
+-- CCiscoPhoneDevSpecificDataPassThrough
```

## Class Detail

```
class CCiscoPhoneDevSpecificDataPassThrough : public CCiscoPhoneDevSpecific
{
public:
    CCiscoPhoneDevSpecificDataPassThrough () :
        CCiscoPhoneDevSpecific(CPDST_DEVICE_DATA_PASSTHROUGH_REQUEST)
    {
        memset((char*)m_DeviceData, 0, sizeof(m_DeviceData)) ;
    }
    virtual ~CCiscoPhoneDevSpecificDataPassThrough() {}
    // data size determined by MAX_DEVICE_DATA_PASSTHROUGH_SIZE
    TCHAR m_DeviceData[MAX_DEVICE_DATA_PASSTHROUGH_SIZE] ;
    // subtract out the virtual function table pointer size
    virtual DWORD dwSize (void) const {return (sizeof (*this)-4) ;}
}
```

## Parameters

DWORD m\_MsgType

Equals CPDST\_DEVICE\_DATA\_PASSTHROUGH\_REQUEST.

DWORD m\_DeviceData

This character buffer contains the XML data that is to be sent to phone device.



### Note

Be aware that MAX\_DEVICE\_DATA\_PASSTHROUGH\_SIZE = 2000.

A phone can pass data to an application and it can get retrieved by using PhoneGetStatus (PHONESTATUS:devSpecificData). See PHONESTATUS description for further details.

## Set Status Msgs

PhoneDevSpecificSetStatusMsgs allows the application to set status bit map to enable specific DEVICE\_DEVSPECIFIC messages to be sent to the application.

The application must open a TAPI phone device with minimum extension version 0x00030000 to use this feature.



### Note

Be aware that this extension requires applications to negotiate extension version as 0x00030000.

```
CCiscoPhoneDevSpecific
|
+-- CCiscoPhoneDevSpecificSetStatusMsgs
```

## Class Detail

```
class CCiscoPhoneDevSpecificSetStatusMsgs:public CCiscoPhoneDevSpecific
{
public:
    CCiscoPhoneDevSpecificSetStatusMsgs() :
        CCiscoPhoneDevSpecific (CPDST_SET_DEVICE_STATUS_MESSAGES) {}
    virtual ~CCiscoPhoneDevSpecificSetStatusMsgs() {}
    DWORD m_DevSpecificStatusMsgFlags ; // PHONE_DEVSPECIFIC
```

```

        // subtract virtual function table pointer
        virtual DWORD dwSize(void) const {return (sizeof (*this) - 4) ; }
    } ;

```

## Parameters

DWORD m\_MsgType

equals CPDST\_SET\_DEVICE\_STATUS\_MESSAGES.

DWORD m\_DevSpecificStatusMsgFlags

Bit map of PHONE\_DEVSPECIFIC event flag

const DWORD DEVSPECIFIC\_DEVICE\_DATA\_PASSTHROUGH\_EVENT = 0x00000001;

const DWORD DEVSPECIFIC\_DEVICE\_SOFTKEY\_PRESSED\_EVENT = 0x00000002;

const DWORD DEVSPECIFIC\_DEVICE\_STATE\_EVENT = 0x00000004;

const DWORD DEVSPECIFIC\_DEVICE\_PROPERTY\_CHANGED\_EVENT = 0x00000008;

## Set Unicode Display

PhoneDevSpecificSetUnicodeDisplay sets the Unicode display on the phone.

The application must open a TAPI phone device with minimum extension version 0x00060000 to use this feature.



### Note

Be aware that this extension requires applications to negotiate extension version as 0x00060000.

```

CCiscoPhoneDevSpecific
|
+-- CCiscoPhoneDevSpecificSetUnicodeDisplay

```

## Class Detail

```

{
    public:
        CCiscoPhoneDevSpecificSetUnicodeDisplay() :
        CCiscoPhoneDevSpecific (CPDST_SET_DEVICE_UNICODE_DISPLAY) {};
        virtual ~CCiscoPhoneDevSpecificSetUnicodeDisplay() {};
        DWORD dwRow;
            DWORD dwColumn;
            DWORD dwSizeOfUnicodeStr;
            wchar_t UnicodeDisplay[MAX_UNICODE_DISPLAY_STRING];
            // subtract virtual function table pointer
            virtual DWORD dwSize(void) const {return (sizeof (*this) - 4) ; }
    } ;

```

## Parameters

DWORD m\_MsgType

Equals CPDST\_SET\_DEVICE\_UNICODE\_DISPLAY.

DWORD m\_dwRow

Row number on the phone display where the Unicode string must be displayed  
 DWORD m\_ dwColumn  
 Column number on the phone display where the Unicode string must be displayed  
 DWORD m\_ dwSizeOfUnicodeStr  
 Size of the Unicode string  
 wchar\_t UnicodeDisplay[MAX\_UNICODE\_DISPLAY\_STRING];  
 Unicode display string, with maximum size of MAX\_UNICODE\_DISPLAY\_STRING  
 MAX\_UNICODE\_DISPLAY\_STRING = 100

## Explicit Acquire

The CCiscoPhoneDevSpecificAcquire class gets used to explicitly acquire any CTI controllable device. If a Super-provider application needs to open any CTI-controllable device on the Cisco Unified Communications Manager system, the application should explicitly acquire that device by using the preceding interface. After successful response, it can open the device as usual.



### Note

Be aware that this extension is only available if extension version 0x00070000 or higher is negotiated.

```
CCiscoPhoneDevSpecific
|
+-- CCiscoPhoneDevSpecificAcquire
```

## Class Details

```
class CCiscoPhoneDevSpecific Acquire : public CCiscoPhoneDevSpecific
{
    public:
    CCiscoPhoneDevSpecificAcquire () : CCiscoPhoneDevSpecific (CPDST_ACQUIRE) {}
    virtual ~ CCiscoPhoneDevSpecificAcquire () {}
    char m_DeviceName[16];
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    // subtract out the virtual function table pointer
};
```

## Parameters

DWORD m\_MsgType  
 Equals CPDST\_ACQUIRE  
 char m\_DeviceName[16]  
 The DeviceName that needs to be explicitly acquired.

## Explicit Deacquire

The CCiscoPhoneDevSpecificDeacquire class gets used to explicitly de-acquire an explicitly acquired device.

If a SuperProvider application explicitly acquired any CTI-controllable device on the Unified Communications Manager system, the application should explicitly de-acquire that device by using this interface.



### Note

Be aware that this extension is only available if extension version 0x00070000 or higher is negotiated.

```
CCiscoPhoneDevSpecific
|
+-- CCiscoPhoneDevSpecificDeacquire
```

## Class Details

```
class CCiscoPhoneDevSpecificDeacquire : public CCiscoPhoneDevSpecific
{
    public:
    CCiscoPhoneDevSpecificDeacquire () : CCiscoPhoneDevSpecific (CPDST_ACQUIRE) {}
        virtual ~ CCiscoPhoneDevSpecificDeacquire () {}
        char m_DeviceName[16];
        virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
        // subtract out the virtual function table pointer
};
```

## Parameters

DWORD m\_MsgType

Equals CPDST\_DEACQUIRE

char m\_DeviceName[16]

The DeviceName that needs to be explicitly de-acquired.

## Request Call RTP Snapshot

The CCiscoPhoneDevSpecificGetRTPSnapshot class gets used to request Call RTP snapshot event from the device. There will be LineCallDevSpecific event for each call on the device.



### Note

Be aware that this extension is only available if extension version 0x00070000 or higher is negotiated.

```
CCiscoPhoneDevSpecific
|
+-- CCiscoPhoneDevSpecificGetRTPSnapshot
```

## Class Details

```
class CCiscoPhoneDevSpecificGetRTPSnapshot: public CCiscoPhoneDevSpecific
{
    public:
```

```

CCiscoPhoneDevSpecificGetRTPSnapshot () : CCiscoPhoneDevSpecific
(CPDST_REQUEST_RTP_SNAPSHOT_INFO) {}
    virtual ~ CCiscoPhoneDevSpecificGetRTPSnapshot () {}
    char m_DeviceName[16];
    virtual DWORD dwSize(void) const {return sizeof(*this)-4;}
    // subtract out the virtual function table pointer
};

```

## Parameters

DWORD m\_MsgType

Equals CPDST\_DEACQUIRE

char m\_DeviceName[16]

The DeviceName that needs to be explicitly de-acquired.

## Messages

This section describes the line device specific messages that the Cisco Unified TSP supports. An application receives nonstandard TAPI messages in the following LINE\_DEVSPECIFIC messages:

- A message to signal when to stop and start streaming RTP audio.
- A message that contains the call handle of active calls when the application starts up.
- A message that indicates to set the RTP parameters based on the data of the message.
- A message that indicates secure media status.

The message type represents an enumerated integer with the following values:

```

enum CiscoLineDevSpecificMsgType
{
    SLDSMT_START_TRANSMISION = 1,
    SLDSMT_STOP_TRANSMISION,
    SLDSMT_START_RECEPTION,
    SLDSMT_STOP_RECEPTION,
    SLDST_LINE_EXISTING_CALL,
    SLDSMT_OPEN_LOGICAL_CHANNEL,
    SLDSMT_CALL_TONE_CHANGED,
    SLDSMT_LINECALLINFO_DEVSPECIFICDATA,
    SLDSMT_HOLD_REVERSION,
    SLDSMT_LINE_PROPERTY_CHANGED,
    SLDSMT_MONITORING_STARTED,
    SLDSMT_MONITORING_ENDED,
    SLDSMT_RECORDING_STARTED,
    SLDSMT_RECORDING_ENDED,
    SLDSMT_NUM_TYPE,
    SLDSMT_IP_ADDRESSING_MODE_CHANGED,
    SLDSMT_START_TRANSMISION_ADDRESSING_MODE,
    SLDSMT_START_RECEPTION_ADDRESSING_MODE,
    SLDSMT_DEVICE_STATE,
    SLDSMT_MONITORING_TERMINATED,
    SLDSMT_MEDIA_TO_BIB_STARTED,
    SLDSMT_MEDIA_TO_BIB_ENDED,
    SLDSMT_MONITORING_MODE_UPDATED,
    SLDSMT_RTP_GET_IP_PORT
};

```

## Start Transmission Events

### SLDSMT\_START\_TRANSMISSION

When a message is received, the RTP stream transmission starts and:

- dwParam2 specifies the network byte order IP address of the remote machine to which the RTP stream should be directed.
- dwParam3, specifies the high-order word that is the network byte order IP port of the remote machine to which the RTP stream should be directed.
- dwParam3, specifies the low-order word that is the packet size, in milliseconds, to use.

The application receives these messages to signal when to start streaming RTP audio. At extension version 1.0 (0x00010000), the parameters have the following format:

- dwParam1 contains the message type.
- dwParam2 contains the IP address of the remote machine.
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.

At extension version 2.0 (0x00020000), start transmission uses the following format:

- dwParam1:from highest order bit to lowest
  - First two bits blank
  - Precedence value 3 bits
  - Maximum frames per packet 8 bits
  - G723 bit rate 2 bits
  - Silence suppression value 1 bit
  - Compression type 8 bits
  - Message type 8 bits
- dwParam2 contains the IP address of the remote machine
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.

At extension version 4.0 (0x00040000), start transmission has the following format:

- hCall – The call of the Start Transmission event
- dwParam1:from highest order bit to lowest
  - First two bits blank
  - Precedence value 3 bits
  - Maximum frames per packet 8 bits
  - G723 bit rate 2 bits
  - Silence suppression value 1 bit
  - Compression type 8 bits
  - Message type 8 bits
- dwParam2 contains the IP address of the remote machine
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.



## Start Reception Events

### SLDSMT\_START\_RECEPTION

When a message is received, the RTP stream reception starts and:

- dwParam2 specifies the network byte order IP address of the local machine to use.
- dwParam3, specifies the high-order word that is the network byte order IP port to use.
- dwParam3, specifies the low-order high-order word that is the packet size, in milliseconds, to use.

When a message is received, the RTP stream reception should commence.

At extension version 1, the parameters have the following format:

- dwParam1 contains the message type.
- dwParam2 contains the IP address of the remote machine.
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.

At extension version 2 start reception uses the following format:

- dwParam1:from highest order bit to lowest
  - First 13 bits blank
  - G723 bit rate 2 bits
  - Silence suppression value 1 bit
  - Compression type 8 bits
  - Message type 8 bits
- dwParam2 contains the IP address of the remote machine
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.

At extension version 4.0 (0x00040000), start reception uses the following format:

- hCall – The call of the Start Reception event
- dwParam1:from highest order bit to lowest
  - First 13 bits blank
  - G723 bit rate 2 bits
  - Silence suppression value 1 bit
  - Compression type 8 bits
  - Message type 8 bits
- dwParam2 contains the IP address of the remote machine
- dwParam3 contains the network byte order IP port of the remote machine to which the RTP stream should be directed in the high-order word and the packet size in milliseconds in the low-order word.

## Stop Transmission Events

### SLDSMT\_STOP\_TRANSMISSION

When a message is received, transmission of the streaming should stop.

At extension version 1.0 (0x00010000), stop transmission uses the following format:

- dwParam1 – Message type

At extension version 4.0 (0x00040000), stop transmission uses the following format:

- hCall – The call for which the Stop Transmission event applies.
- dwParam1 – Message type

## Stop Reception Events

### SLDSMT\_STOP\_RECEPTION

When a message is received, reception of the streaming should stop.

At extension version 1.0 (0x00010000), stop reception uses the following format:

- dwParam1 - message type

At extension version 4.0 (0x00040000), stop reception uses the following format:

- hCall – The call for which the Stop Reception event applies.
- dwParam1 – Message type

## Existing Call Events

### SLDST\_LINE\_EXISTING\_CALL

These events inform the application of existing calls in the PBX when it starts up. The format of the parameters follows:

- dwParam1 – Message type
- dwParam2 – Call object
- dwParam3 – TAPI call handle

## Open Logical Channel Events

### SLDSMT\_OPEN\_LOGICAL\_CHANNEL

When a call has media established at a CTI Port or Route Point that is registered for Dynamic Port Registration, receipt of this message indicates that an IP address and UDP port number need to be set for the call.



#### Note

---

This extension is only available if extension version 0x00040000 or higher gets negotiated.

---

The following format of the parameters applies:

- hCall - The call for which the Open Logical Channel event applies

- dwParam1 – Message type
- dwParam2 – Compression Type
- dwParam3 – Packet size in milliseconds

At extension version 9.0 (0x00090000), start transmission has the following format:

- hCall - The call the Open Logical Channel event is for
- dwParam1: from highest order bit to lowest
- First eight bits blank
- Maximum frames per packet 8 bits
- Compression type 8 bits
- Message type 8 bits
- dwParam2 contains the IP addressing mode
- dwParam3: Packet size in milliseconds

At extension version B.0 (0x000B0000), Open Logical channel has the following format:

- hCall - The call the Open Logical Channel event is for
- dwParam1: from highest order bit to lowest
- First sixteen bits blank
- Compression type 8 bits
- Message type 8 bits
- dwParam2: from highest order bit to lowest
- First twenty three bits blank
- SetRTPIInfo (twenty fourth bit from MSB/ninth bit from LSB)
- IP addressing mode 8 bits
- dwParam3: Packet size in milliseconds

## LINECALLINFO\_DEVSPECIFICDATA Events

### SLDSMT\_LINECALLINFO\_DEVSPECIFICDATA

This message indicates that the DEVSPECIFICDATA information is changed in the DEVSPECIFIC portion of the LINECALLINFO structure for the different fields.



#### Note

The fields are only available if the negotiated version contains support for the particular feature.

The following format applies for the parameters:

- hCall - The call handle
- dwParam1 - Message type
- dwParam2

```
SLDST_SRTP_INFO | SLDST_QOS_INFO | SLDST_PARTITION_INFO | SLDST_EXTENDED_CALL_INFO |
SLDST_CALL_ATTRIBUTE_INFO | SLDST_CCM_CALLID | SLDST_CALL_SECURITY_STATUS |
SLDST_NUMBER_TYPE_CHANGED | SLDST_GLOBALIZED_CALLING_PARTY_CHANGED |
SLDST_FAR_END_IP_ADDRESS_CHANGED | SLDST_UNIQUE_CALL_REF_ID_INFO
```

The bit mask values follow:

```
SLDST_SRTP_INFO      0x00000001
SLDST_QOS_INFO      0x00000002
SLDST_PARTITION_INFO 0x00000004
SLDST_EXTENDED_CALL_INFO 0x00000008
SLDST_CALL_ATTRIBUTE_INFO 0x00000010
SLDST_CCM_CALL_ID 0x00000020
SLDST_SECURITY_STATUS_INFO 0x00000040
SLDST_NUMBER_TYPE_CHANGED 0x00000080
SLDST_GLOBALIZED_CALLING_PARTY_CHANGED 0x00000100
SLDST_FAR_END_IP_ADDRESS_CHANGED 0x00000200
SLDST_UNIQUE_CALL_REF_ID_INFO 0x00000400
```

For example, if there are changes in SRTP and QoS but not in Partition, then both the SLDST\_SRTP\_INFO and SLDST\_QOS\_INFO bits are set. The value for dwParam2 = SLDST\_SRTP\_INFO | SLDST\_QOS\_INFO = 0x00000011

- dwParam3 - If a change occurs in the SRTP information, then this field contains the CiscoSecurityIndicator.

```
enum CiscoSecurityIndicator
{
    SRTP_MEDIA_ENCRYPT_KEYS_AVAILABLE,
    SRTP_MEDIA_ENCRYPT_USER_NOT_AUTH,
    SRTP_MEDIA_ENCRYPT_KEYS_UNAVAILABLE,
    SRTP_MEDIA_NOT_ENCRYPTED
};
```



**Note**

dwParam3 is used when dwParam2 has the SRTP bit mask set.

## Call Tone Changed Events

### SLDSMT\_CALL\_TONE\_CHANGED

When a tone change occurs on a call, receipt of this message indicates the tone and the feature that caused the tone change.



**Note**

Be aware that this extension is only available if extension version 0x00050000 or higher is negotiated. In the Cisco Unified TSP 4.1 release and later, this event only gets sent for Call Tone Changed Events where the tone equals CTONE\_ZIPZIP and the tone gets generated as a result of the FAC/CMC feature.

The format of the parameters follows:

- hCall—The call for which the Call Tone Changed event applies
- dwParam—Message type
- dwParam2—CTONE\_ZIPZIP, 0x31 (Zip Zip tone), CTONE\_ZIP, 0x32 (Zip tone)
- dwParam3—If dwParam2 is CTONE\_ZIPZIP, this parameter contains a bitmask with the following possible values:
  - CZIPZIP\_FACREQUIRED—If this bit is set, it indicates that a FAC is required.
  - CZIPZIP\_CMCREQUIRED—If this bit is set, it indicates that a CMC is required.
  - -If dwParam2 is CTONE\_ZIP, this parameter contains direction mode with the following possible values:

- 0 - Tone is played at local End
- 1 - Tone is played at Remote End

**Note**

For a DN that requires both codes, the first event always applies for the FAC and CMC code. The application optionally can send both codes separated by # in the same request. The second event generation remains optional based on what the application sends in the first request.

## Line Property Changed Events

### SLDSMT\_LINE\_PROPERTY\_CHANGED

When a line property is changed, a LINEDEVSPECIFIC event is fired with indication of the changes.

**Note**

This extension is available only if extension version 0x00080000 or higher is negotiated.

The format of the parameters follows:

dwParam1 - Message type

dwParam2 - indication type - CiscoLinePropertyChangeType

```
enum CiscoLinePropertyChangeType
{
    LPCT_INTERCOM_LINE           = 0x00000001,
    LPCT_RECORDING_TYPE         = 0x00000002,
    LPCT_MAX_CALLS              = 0x00000004,
    LPCT_BUSY_TRIGGER           = 0x00000008,
    LPCT_LINE_INSTANCE          = 0x00000010,
    LPCT_LINE_LABEL              = 0x00000020,
    LPCT_VOICEMAIL_PILOT        = 0x00000040,
    LPCT_DEVICE_IPADDRESS        = 0x00000080,
    LPCT_NEWCALL_ROLLOVER        = 0x00000100,
    LPCT_CONSULTCALL_ROLLOVER    = 0x00000200,
    LPCT_JOIN_ON_SAME_LINE       = 0x00000400,
    LPCT_JOIN_ACROSS_LINE        = 0x00000800,
    LPCT_DIRECTTRANSFER_ON_SAME_LINE = 0x00001000,
    LPCT_DIRECTTRANSFER_ACROSS_LINE = 0x00002000
};
```

dwParam3 - default = 0,

In case, dwParam2 = LPCT\_INTERCOM\_LINE, dwParam3 is the result of the change

```
Enum CiscoIntercomLineChangeResult
{
    IntercomSettingChange_successful = 0;
    IntercomSettingRestorationFail = 1
}
```

If dwParam2 = LPCT\_RECORDING\_TYPE, dwParam3 will have a new Recording Type:

```
enum recordType
{
    RecordType_NoRecording = 0,
    RecordType_AutomaticRecording = 1,
    RecordType_ApplicationInvokedCallRecording = 2,
    RecordType_DeviceInvokedCallRecording = 3
};
```

## Monitoring Started Event

### SLDSMT\_MONITORING\_STARTED

When monitoring starts on a particular call, this event is triggered for the monitored call to inform the application.

**Note**

---

This event is available only if extension version 0x00080000 or higher is negotiated.

---

The format of the parameters follows:

- dwParam1—Message type
- dwParam2—0
- dwParam3—0

## Monitoring Ended Event

### SLDSMT\_MONITORING\_ENDED

When monitoring is stopped for a particular call, this event is triggered for the monitored call to inform the application.

**Note**

---

This event is available only if extension version 0x00080000 or higher is negotiated.

---

The format of the parameters follows:

- dwParam1—Message type
- dwParam2—Reason code
- dwParam3—0

## Recording Started Event

### SLDSMT\_RECORDING\_STARTED

When recording starts on a particular call, this event is triggered to inform the same to the application.

**Note**

---

This event is available only if extension version 0x00080000 or higher is negotiated.

---

The format of the parameters follows:

- dwParam1—Message type
- dwParam2—0
- dwParam3—0

## Recording Ended Event

### SLDSMT\_RECORDING\_ENDED

When recording is stopped on a particular call, this event is triggered to inform the same to the application.

**Note**

---

This event is available only if extension version 0x00080000 or higher is negotiated.

---

The format of the parameters follows:

- dwParam1—Message type
- dwParam2—Reason code
- dwParam3—0

## Silent Monitoring Session Terminated Event

### SLDSMT\_MONITORING\_TERMINATED

When Monitoring Session is toned down as security capabilities of the supervisor do not meet or exceed the capabilities of agent, this event is fired on the supervisor to inform the same to the application.

**Note**

---

This event is only available if extension version 0x000A0000 or higher is negotiated.

---

The format of the parameters follows:

- dwParam1 – Message type - SLDSMT\_MONITORING\_TERMINATED
- dwParam2 – TransactionID – which is unique for the Monitoring session
- dwParam3 – New Cause Code - LINEDISCONNECTMODE\_INCOMPATIBLE

## Media To BIB Started Event

### SLDSMT\_MEDIA\_TO\_BIB\_STARTED

This event indicates that agent greeting call has been successfully set up.

**Note**

---

This event is only available if extension version 0x000B0000 or higher is negotiated.

---

The format of the parameters follows:

- dwParam1 – Message type - SLDSMT\_MEDIA\_TO\_BIB\_STARTED
- dwParam2 – reserved (0)
- dwParam3 – reserved (0)

## Media To BIB Ended Event

### SLDSMT\_MEDIA\_TO\_BIB\_ENDED

This event indicates that the agent greeting has ended.

**Note**

---

This event is only available if extension version 0x000B0000 or higher is negotiated.

---

The format of the parameters follows:

- dwParam1 – Message type - SLDSMT\_MEDIA\_TO\_BIB\_ENDED
- dwParam2 – result code:
  - non 0: Agent Greeting was successfully played
  - 0: Agent Greeting was not successfully played
- dwParam3 – result code

## Get IP and Port Event

### SLDSMT\_RTP\_GET\_IP\_PORT

This event indicates that the application has to set the RTP Port and IP information using existing SetRTP devspecific Extension. The application has to set the RTP information only for Dynamically Registered CTI Ports or Route Points and for static Registered CTI Ports, application has to open the port used for registration.

**Note**

---

This event is available only if extension version 0x000B0000 or higher is negotiated.

---

The format of the parameters follows:

- dwParam1 – Message type - SLDSMT\_RTP\_GET\_IP\_PORT
- dwParam2 – IP Addressing Capability (from highest order bit to lowest)
  - First twenty three bits blank
  - SetRTPInfo (twenty fourth bit from MSB or ninth bit from LSB)
  - IP addressing mode 8 bits
- dwParam3 – reserved (0)

## Monitor Mode Update Event

### SLDSMT\_MONITORING\_MODE\_UPDATED

This event indicates that the monitoring mode has been successfully updated to the value in dwParam1 and is sent to active supervisor and agent lines.

**Note**

---

This event is available only if extension version 0x000B0000 or higher is negotiated.

---

The format of the parameters follows:

- dwParam1 – Message type - SLDSMT\_MONITORING\_MODE\_UPDATED



- dwParam2 – monitoring mode

```
enum
{
    MonitorMode_Silent = 1,
    MonitorMode_Whisper = 2,
    MonitorMode_Active = 3    // Not currently used
} MonitorMode;
```

- dwParam3 – active tone direction

```
enum
{
    PlayToneDirection_NoLocalOrRemote = 0,
    PlayToneDirection_LocalOnly,
    PlayToneDirection_RemoteOnly,
    PlayToneDirection_BothLocalAndRemote
} PlayToneDirection
```

