



開発者用 SIP 透過性および正規化ガイド

Developer Guide for SIP Transparency and Normalization

Cisco Unified Communications Manager Release 8.5

**【注意】シスコ製品をご使用になる前に、安全上の注意
(www.cisco.com/jp/go/safety_warning/)をご確認ください。**

本書は、米国シスコシステムズ発行ドキュメントの参考和訳です。
リンク情報につきましては、日本語版掲載時点で、英語版にアップ
デートがあり、リンク先のページが移動/変更されている場合があ
りますことをご了承ください。
あくまでも参考和訳となりますので、正式な内容については米国サ
イトのドキュメントを参照ください。

また、契約等の記述については、弊社販売パートナー、または、弊
社担当者にご確認ください。

このマニュアルに記載されている仕様および製品に関する情報は、予告なしに変更されることがあります。このマニュアルに記載されている表現、情報、および推奨事項は、すべて正確であると考えていますが、明示的であれ黙示的であれ、一切の保証の責任を負わないものとします。このマニュアルに記載されている製品の使用は、すべてユーザ側の責任になります。

対象製品のソフトウェア ライセンスおよび限定保証は、製品に添付された『Information Packet』に記載されています。添付されていない場合には、代理店にご連絡ください。

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

ここに記載されている他のいかなる保証にもよらず、各社のすべてのマニュアルおよびソフトウェアは、障害も含めて「現状のまま」として提供されます。シスコシステムズおよびこれら各社は、商品性の保証、特定目的への準拠の保証、および権利を侵害しないことに関する保証、あるいは取引過程、使用、取引慣行によって発生する保証をはじめとする、明示されたまたは黙示された一切の保証の責任を負わないものとします。

いかなる場合においても、シスコシステムズおよびその供給者は、このマニュアルの使用または使用できないことによって発生する利益の損失やデータの損傷をはじめとする、間接的、派生的、偶発的、あるいは特殊な損害について、あらゆる可能性がシスコシステムズまたはその供給者に知らされていても、それらに対する責任は一切負わないものとします。

Cisco and the Cisco Logo are trademarks of Cisco Systems, Inc. and/or its affiliates in the U.S. and other countries. A listing of Cisco's trademarks can be found at www.cisco.com/go/trademarks. Third party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1005R)

このマニュアルで使用している IP アドレスおよび電話番号は、実際のアドレスおよび電話番号を示すものではありません。マニュアル内の例、コマンド出力、ネットワーク トポロジ図、およびその他の図は、説明のみを目的として使用されています。説明の中に実際のアドレスおよび電話番号が使用されていたとしても、それは意図的なものではなく、偶然の一致によるものです。

開発者用 SIP 透過性および正規化ガイド Cisco Unified Communications Manager 8.5
© 2010 Cisco Systems, Inc.

All rights reserved.

Copyright © 2010–2011, シスコシステムズ合同会社.
All rights reserved.



CONTENTS

はじめに	vii
対象読者	vii
マニュアルの構成	vii
表記法	viii
マニュアルの入手方法およびテクニカル サポート	ix
開発者のサポート	ix

CHAPTER 1

概要	1-1
概要	1-1
スクリプト環境	1-2
メッセージ ハンドラ	1-4
命名	1-4
ワイルドカード	1-5
メッセージ ハンドラを取得するルール	1-6

CHAPTER 2

SIP と SDP の正規化	2-1
----------------	-----

CHAPTER 3

SIP メッセージの API	3-1
要求行または応答行での操作	3-1
getRequestLine	3-1
getRequestUriParameter	3-2
setRequestUri	3-3
getResponseLine	3-3
setResponseCode	3-4
ヘッダーの操作	3-4
allowHeader	3-5
getHeader	3-5
getHeaderValues	3-6
getHeaderValueParameter	3-7
getHeaderUriParameter	3-7
addHeader	3-8
addHeaderValueParameter	3-8
addHeaderUriParameter	3-9
modifyHeader	3-10

applyNumberMask	3-10
番号マスクの適用	3-11
convertDiversionToHi	3-11
convertHiToDiversion	3-12
removeHeader	3-13
removeHeaderValue	3-13
SDP の操作	3-14
getSdp	3-14
setSdp	3-15
removeUnreliableSdp	3-16
Content 本体の操作	3-17
getContentBody	3-17
addContentBody	3-18
removeContentBody	3-19
メッセージのブロック	3-19
block	3-19
透過性	3-20
getPassThrough	3-20
Per-Dialog コンテキストの管理	3-20
getContext	3-20
ユーティリティ	3-22
isInitialInviteRequest	3-22
isRelInviteRequest	3-23
getUri	3-24

CHAPTER 4

SDP API 4-1

行レベルの API	4-2
Cisco API より	4-2
getLine	4-3
modifyLine	4-3
addline	4-5
insertLineAfter	4-6
insertLineBefore	4-7
removeLine	4-8
メディア記述レベルの API	4-9
getMediaDescription(media-level)	4-11
getMediaDescription(media-contains)	4-12
modifyMediaDescription(media-level, media-description)	4-13
modifyMediaDescription(media-contains, media-description)	4-14

addMediaDescription(media-description)	4-15
insertMediaDescription(media-level, media-description)	4-17
removeMediaDescription(media-level)	4-18
removeMediaDescription(media-contains)	4-18

CHAPTER 5	SIP パススルー API	5-1
	addHeader	5-1
	addHeaderValueParameter	5-2
	addHeaderUriParameter	5-3
	addRequestUriParameter	5-4
	addContentBody	5-4

CHAPTER 6	SIP ユーティリティ API	6-1
	parseUri	6-1

CHAPTER 7	SIP URI API	7-1
	applyNumberMask	7-1
	getHost	7-2
	getUser	7-3
	encode	7-3

CHAPTER 8	トレース API	8-1
	trace.format	8-1
	trace.enable	8-2
	trace.disable	8-2
	trace.enabled	8-2

CHAPTER 9	スクリプトパラメータ API	9-1
	getValue	9-1

CHAPTER 10	SIP 透過性	10-1
	サポートされる機能	10-1
	181 透過性の例	10-4
	INFO 透過性の例	10-6



はじめに

このマニュアルでは、Cisco Unified CM- Session Manager Edition (CUCM-SME) での SIP メッセージのカスタマイズプロセスについて説明します。また、CUCM-SME および API で使用可能な、SIP の透過性機能および正規化機能をサポートするための Lua 環境の詳細についても説明します。

「はじめに」は、次の内容で構成されています。

- 「対象読者」
- 「マニュアルの構成」
- 「表記法」
- 「マニュアルの入手方法およびテクニカル サポート」
- 「開発者のサポート」

対象読者

このマニュアルでは、Cisco Unified Communications Manager 8.5(1) と統合されたアプリケーションまたは製品を SIP の透過性機能および正規化機能を使用して開発する開発者、ベンダ、およびカスタマーを対象とした情報を提供します。

マニュアルの構成

このマニュアルは、次の章で構成されています。

章	説明
第 1 章「概要」	個別の配置で Session Management (SM; セッション管理) の動作をカスタマイズするために使用されるインターフェイスである Lua 環境の概要を説明します。
第 2 章「SIP と SDP の正規化」	SIP メッセージおよびそれに関連するあらゆる Session Description Protocol (SDP; セッション記述プロトコル) を扱うための Lua スクリプト作成環境 API を紹介します。
第 3 章「SIP メッセージの API」	メッセージの処理を可能にする Lua スクリプト作成環境 SIP Message API について説明します。
第 4 章「SDP API」	Session Description Protocol (SDP; セッション記述プロトコル) のコンテンツ本文に関連する API について説明します。

章	説明
第 5 章「SIP パススルー API」	情報を 1 つのコール レッグから別のコール レッグへと受け渡すことを可能にするパススルー オブジェクト提供 API について説明します。
第 6 章「SIP ユーティリティ API」	データ文字列の処理を可能にする API について説明します。
第 7 章「SIP URI API」	解析済み SIP URI の処理を可能にする API について説明します。
第 8 章「トレース API」	トレーシングによって、スクリプト作成者がどのようにスクリプト内からトレースを生成できるようになるのかについて説明します。これは、スクリプトのデバッグだけに使用してください。
第 9 章「スクリプト パラメータ API」	スクリプト パラメータによって、スクリプト作成者がどのようにトランク固有の設定パラメータ値を取得できるようになるのかについて説明します。
第 10 章「SIP 透過性」	SIP 透過性で使用される Lua スクリプト作成環境 API を紹介します。

表記法

このマニュアルでは、次の表記法を使用しています。

表記法	説明
boldface フォント	コマンドおよびキーワードは 太字 で示しています。
<i>italic</i> フォント	ユーザが値を指定する引数は、 <i>イタリック体</i> で示しています。
[]	角カッコの中の要素は、省略可能です。
{ x y z }	必ずどれか 1 つを選択しなければならない必須キーワードは、波カッコで囲み、縦棒で区切って示しています。
[x y z]	どれか 1 つを選択できる省略可能なキーワードは、角カッコで囲み、縦棒で区切って示しています。
string	引用符を付けない一組の文字。 string の前後には引用符を使用しません。引用符を使用すると、その引用符も含めて string とみなされます。
screen フォント	システムが表示する端末セッションおよび情報は、 screen フォントで示しています。
boldface screen フォント	ユーザが入力しなければならない情報は、 太字の screen フォントで示しています。
<i>italic screen</i> フォント	ユーザが値を指定する引数は、 <i>イタリック体の screen</i> フォントで示しています。
→	この矢印は、例の中の重要な行やテキストを強調するためのものです。
^	^ 記号は、Ctrl キーを表します。たとえば、画面に表示される ^D というキーの組み合わせは、Ctrl キーを押しながら D キーを押すことを意味します。
< >	パスワードのように出力されない文字は、山カッコ (<>) で囲んで示しています。

(注) は、次のように表しています。



(注)

「注釈」です。役立つ情報や、このマニュアル以外の参照資料などを紹介しています。



注意

「要注意」の意味です。機器の損傷またはデータ損失を予防するための注意事項が記述されています。



ヒント

次の情報が問題の解決に役立つ場合があるという意味です。

マニュアルの入手方法およびテクニカル サポート

マニュアルの入手方法、テクニカル サポート、その他の有用な情報について、次の URL で、毎月更新される『What's New in Cisco Product Documentation』を参照してください。シスコの新規および改訂版の技術マニュアルの一覧も示されています。

<http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html>

『What's New in Cisco Product Documentation』は RSS フィードとして購読できます。また、リーダーアプリケーションを使用してコンテンツがデスクトップに直接配信されるように設定することもできます。RSS フィードは無料のサービスです。シスコは現在、RSS バージョン 2.0 をサポートしています。

開発者のサポート

Developer Support Program は、Cisco Systems インターフェイスを公式にサポートするもので、Cisco Service Provider のソリューションである Ecosystem および Cisco Partner プログラムの開発者、カスタマー、およびパートナーが相互にソリューションを提供できるようにします。

製品テクノロジー エンジニアリング チームの拡張である Developer Support Engineers は、専門的なサポートをタイムリーに提供するために必要になるリソースに直接アクセスできます。

このプログラムの詳細については、Developer Support Program の Web サイト (<http://www.cisco.com/go/developer support/>) を参照してください。

SIP Line Messaging Guide を使用する開発者の方は、Cisco Developer Support Program に加入することをお勧めします。この新しいプログラムでは、開発プロジェクトでシスコのインターフェイスをご利用の際に、一貫したレベルのサポートを提供します。

Cisco Technical Assistance Center (TAC) のサポートには、SIP の開発者向けサポートは含まれていません。また、サポートの範囲は、Cisco 製品のインストールと設定、ならびにシスコ製アプリケーションに限定されています。Cisco Developer Support Program の詳細については、developer-support@cisco.com までお問い合わせください。



CHAPTER 1

概要

このマニュアルでは、Cisco Unified CM- Session Management (SME) の SIP メッセージのカスタマイズに使用されるインターフェイス仕様について説明します。また、Cisco Unified CM-SME および API で使用可能な、SIP の透過性機能および正規化機能をサポートするための Lua 環境の詳細についても説明します。

この章では、次のトピックについて説明します。

- 「概要」
- 「スクリプト環境」
- 「メッセージハンドラ」

概要

この章では、特定の配置環境での、Session Management (SM; セッション管理) の動作のカスタマイズに使用されるインターフェイスである、Lua 環境について説明します。Lua は、一般的な、動作が軽いスクリプト言語です。このマニュアルの対象読者は、Lua スクリプト言語の基礎知識があることを前提としています。

Cisco Unified CM では、SIP の正規化および透過性と呼ばれる機能のセットが提供され、SIP メッセージがカスタマイズされます。

- 正規化：着信メッセージと発信メッセージを変換する処理です。
 - 着信メッセージでは、正規化は、ネットワークからメッセージを受信した後に発生します。着信メッセージの正規化は、メッセージを Cisco Unified CM により適するようにするために、使用されます。たとえば、Cisco Unified CM では、リダイレクト番号情報を伝達するための Diversion ヘッダーがサポートされます。Cisco Unified CM に接続される一部の SIP デバイスでは、この目的で History-Info ヘッダーが使用されます。着信の最適化では、Cisco Unified CM でリダイレクト情報が認識されるよう、History-Info ヘッダーが Diversion ヘッダーに変換されます。
 - 発信メッセージでは、正規化は、ネットワークにメッセージを送信する直前に発生します。したがって、発信メッセージの正規化は、メッセージが外部 SIP デバイス (例：別の SIP 対応 PBX) により適するようにするために使用されます。たとえば、発信の正規化は、外部 SIP デバイスでリダイレクト情報が認識されるよう、Diversion ヘッダーを History-Info ヘッダーに変換するために使用できます。
- 透過性：これを使用すると、Cisco Unified CM が Back to Back User Agent (B2BUA) の場合でも、1 つのコール レッグから別のコール レッグに SIP 情報を渡すことができます。

正規化と透過性の機能は、SIP トランクに関連付けられているスクリプトによって示されます。スクリプトは、発着信 SIP メッセージで動作するメッセージハンドラのセットとして示されます。正規化では、スクリプトによって、次のものを含む SIP メッセージのほとんどの状態が操作されます。

- 要求 URI
- 応答コードとフレーズ
- SIP ヘッダー
- SIP パラメータ
- Content の本体
- SDP

透過性では、スクリプトによって、次のものを含む SIP メッセージのほとんどの情報が渡されます。

- SIP ヘッダー
- SIP パラメータ
- Content の本体

このマニュアルでは、SIP メッセージ情報を操作し、渡すために使用される、スクリプト環境および API について説明します。

スクリプト環境

Cisco Unified CM SIP Trunk の動作をカスタマイズするインターフェイスは、Lua と呼ばれるスクリプト言語によって提供されます。Lua は、オープンソースの、動作が軽いスクリプト言語です。

Cisco Unified CM (SM) で使用できる Lua 環境は、Lua の限定的なサブセットです。Lua では、次のような基本機能が提供されます。

- 単語表記
- 値とタイプ
- 変数
- 文
- 式など

Lua では、次の一部の基本ライブラリも提供されます。

- ベース
- コルーチン
- モジュール
- 文字列操作
- テーブル操作
- 演算関数
- OS 機能
- IO 機能と
- デバッグ機能

ただし、SM で使用可能な Cisco SIP Lua 環境では、ベースライブラリの全体およびサブセットの文字列ライブラリのみがサポートされます。他のライブラリはサポートされません。

ベース ライブラリでは、次のものがサポートされます。

- `ipairs`
- `pairs`
- `next`
- `unpack`
- `error`
- `type`
- `tostring`

Cisco SIP Lua 環境では、スクリプトで使用されるグローバル環境が提供されます。スクリプトに対して、デフォルトの Lua グローバル環境 (`_G` など) は示されません。

Lua スクリプトでは、**メッセージハンドラ**と呼ばれるコールバック機能のセットが提供され、SM 環境のコンテキストで SIP メッセージが操作されます。メッセージハンドラの名前は、特定の SIP メッセージに対して起動されるハンドラを示します。たとえば、スクリプトの「`inbound_INVITE`」メッセージハンドラは、Cisco Unified CM で着信 INVITE を受信したときに、起動されます。メッセージハンドラでは、SIP メッセージを表す `msg` と呼ばれる 1 つのパラメータを受信します。Lua スクリプトでは、Cisco SIP Message ライブラリで定義されている API を使用して、`msg` パラメータが操作されます。

以下では、**メッセージハンドラ**の構造の詳細について説明します。次の項では、Cisco SIP Message ライブラリ API の詳細について説明します。

ここでは、発信 INVITE で「Cisco-Guid」ヘッダーを削除するスクリプトの例を示します。スクリプトの説明を簡単にするため、スクリプトの左側に行番号を示します。

スクリプト例 : M.lua

```
1. M = {}
2. function M.inbound_INVITE(msg)
3.     msg:convertHIToDiversion()
4. end
5. function M.outbound_INVITE(msg)
6.     msg:removeHeader("Cisco-Guid")
7. end
8. return M
```

前述のスクリプトには、3 つの重要な部分があります。

1. モジュールの初期化：スクリプトの最初の行で、「M」と呼ばれる Lua テーブルが作成され、空に初期化されます。このテーブルには、このスクリプトによって提供されるコールバック関数のセットが含まれます。変数 **M** は Lua テーブルで、モジュールの名前でもあります。
2. メッセージハンドラ定義：行 2～4 では、着信 INVITE **メッセージハンドラ**が定義されます。このスクリプトの SIP トランクで着信 INVITE を受信したときに、このコールバック関数が実行されます。この例では、メッセージハンドラによって API が起動され、History-Info ヘッダーが Diversion ヘッダーに変換されます。

行 5～6 では、発信 INVITE **メッセージハンドラ**が定義されます。このスクリプトの SIP トランクで発信 INVITE が送信されたときに、このコールバック関数が実行されます。この例では、メッセージハンドラによって API が起動され、「Cisco-Guid」ヘッダーが削除されます。

スクリプトによって、複数のメッセージハンドラを定義できます。メッセージハンドラの名前によって、指定された SIP メッセージに対して起動されるメッセージハンドラが（ある場合には）示されます。

3. モジュールに戻る：最後の行で、モジュールの名前が返されます。この行は、絶対に必要です。これが、Cisco SIP Lua 環境で、スクリプトに関連付けられているメッセージハンドラが見つけれられる方法です。

メッセージハンドラ

Lua スクリプトでは、メッセージハンドラと呼ばれるコールバック機能のセットが提供され、SM 環境のコンテキストで SIP メッセージが操作されます。メッセージハンドラの名前は、特定の SIP メッセージに対して起動されるハンドラを示します。

命名

メッセージハンドラの命名規則によって、指定された SIP メッセージに対して起動されるメッセージが示されます。仕様による多様な SIP メッセージは、**要求と応答**に分かれます。

- **要求**では、メッセージハンドラは、メッセージの方向と SIP 要求のメソッド名に従って命名されます。メソッド名は、SIP メッセージの「要求行」にあるものです。

```
<direction>_<method>
```

例

```
inbound_INVITE
outbound_UPDATE
```

- **応答**では、メッセージハンドラは、メッセージの方向、応答コード、および SIP メソッドに従って命名されます。応答では、SIP メッセージの **CSeq** ヘッダーから、メソッド名が取得されます。

```
<direction>_<response code>_<method>
```

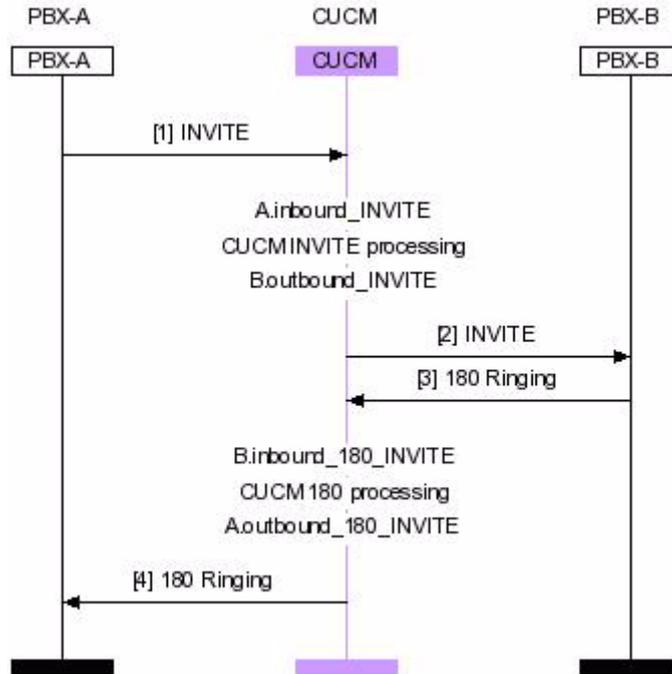
例

```
inbound_183_INVITE
inbound_200_INVITE
outbound_200_UPDATE
```

使用例

Cisco Unified CM-SME が 2 つのトランクを介して PBX-A および PBX-B に接続される場合について考えます。モジュール **A** を返すスクリプトは、PBX-A に接続されているトランクに接続されます。同様に、モジュール **B** を返すスクリプトは、PBX-B に接続されているトランクに接続されます。

次のハンドラが、INVITE ダイアログに対して実行されます。



ワイルドカード

メッセージハンドラでは、サポートされるワイルドカードも命名されます。ワイルドカードのサポートは、メッセージが**要求 SIP**メッセージか**応答 SIP**メッセージかに依存します。

要求メッセージ

ワイルドカード **ANY** は、<method> の場所に使用できます。<direction> では、ワイルドカードはサポートされません。

有効な要求メッセージハンドラ名については、[表 1-1 \(P.1-5\)](#) を参照してください。

表 1-1 有効な要求メッセージハンドラ名

メッセージハンドラ	説明
M.inbound_INVITE	このメッセージハンドラは、初期 INVITE および reINVITE を含むすべての着信 INVITE メッセージに対して起動されます。
M.inbound_ANY	このメッセージハンドラは、すべての着信要求に対して起動されます。
M.outbound_ANY	このメッセージハンドラは、すべての発信要求に対して起動されます。

応答メッセージ

ワイルドカード **ANY** は、<method> と <response code> のいずれか一方または両方の場所に使用できます。<direction> では、ワイルドカードはサポートされません。さらに、ワイルドカード文字 **X** は <response code> で使用できます。

有効な応答メッセージハンドラ名については、[表 1-2 \(P.1-6\)](#) を参照してください。

表 1-2 有効な応答メッセージハンドラ名

メッセージハンドラ	説明
M.inbound_18X_INVITE	このメッセージハンドラは、180、181、182、および 183 を含むすべての着信 18X に対して起動されます。
M.inbound_ANY_INVITE	このメッセージハンドラは、すべての 18X、2XX、3XX、4XX、5XX、および 6XX の応答を含む INVITE 要求のすべての着信応答に対して起動されます。
M.outbound_ANY_ANY	このメッセージハンドラは、要求メソッドに関係なく、すべての発信応答に対して起動されます。

メッセージハンドラを取得するルール

Cisco Unified CM では、次のルールを使用して、該当するメッセージのメッセージハンドラが見つかります。

1. メッセージハンドラでは、大文字と小文字が区別されます。
2. 方向は、着信または発信のいずれかです。
3. 方向は、常に、小文字で表記されます。
4. メッセージの方向は、SM に対する相対方向です。



(注) メッセージの方向は、SIP のダイアログの方向から完全に独立しています。

例

inbound_INVITE は有効なハンドラ名です。InBouND_INVITE は有効なハンドラ名ではありません。

5. SIP メッセージから取得されるメソッド名は、スクリプトで適切なメッセージハンドラを見つける目的で、小文字に変換されます。
6. CUCM-SME では、最長一致基準を使用して、正しいメッセージハンドラが見つかります。

例

スクリプトに、inbound_ANY_ANY と inbound_183_INVITE の 2 つのメッセージハンドラがあるとします。Cisco Unified CM で 183 の応答を受信すると、これは最も明示的な一致のため、inbound_183_INVITE ハンドラが実行されます。



(注) 着信または発信は ANY (応答コード) および ANY (メソッド) でサポートされますが、特定の応答コードでの ANY (メソッド) ワイルドカードは、現在、サポートされません。

つまり、次のメッセージハンドラが有効です。

```
inbound_ANY_ANY
outbound_ANY_ANY
```

しかし、次のものは無効です。

```
inbound_401_ANY
outbound_200_ANY
など
```



CHAPTER 2

SIP と SDP の正規化

Lua スクリプト環境によって、SIP メッセージを操作する付属オブジェクトおよび関連するすべての Session Description Protocol (SDP; セッション記述プロトコル) が確立されます。スクリプトでは、これらのオブジェクトを気にする必要はなく、次の API の次のセットを使用してこれらのオブジェクトにアクセスします。

- 「[SIP メッセージの API](#)」: これらの API を使用すると、スクリプトでは、多様な方法で SIP メッセージを操作できます。
- 「[SDP API](#)」: これらの API を使用すると、スクリプトでは、多様な方法で SDP を操作できます。
- 「[SIP パススルー API](#)」: これらの API を使用すると、スクリプトでは、1 つのコール レッグから別のコール レッグへ情報を渡せます。
- 「[SIP ユーティリティ API](#)」: これらの API により、Uniform Resource Identifier (URI; ユニフォーム リソース識別子) の SIP URI オブジェクトへの解析を含み、データを操作するスクリプトに役に立つユーティリティが提供されます。
- 「[SIP URI API](#)」: これらの API を使用すると、スクリプトでは、解析済み SIP URI オブジェクトを操作できます。
- 「[トレース API](#)」: これらの API を使用すると、スクリプトでは、トレースを有効および無効にし、トレースがイネーブルかどうかを特定し、トレースを生成することができます。
- 「[スクリプト パラメータ API](#)」: このパラメータを使用すると、スクリプト作成者は、トランク固有の設定パラメータ値を取得できます。



CHAPTER 3

SIP メッセージの API

Lua スクリプト環境では、メッセージを操作できる API のセットが提供されます。これらの API について、次の項で説明します。

- 「要求行または応答行での操作」(P.3-1)
- 「ヘッダーの操作」(P.3-4)
- 「SDP の操作」(P.3-14)
- 「Content 本体の操作」(P.3-17)
- 「メッセージのブロック」(P.3-19)
- 「透過性」(P.3-20)
- 「Per-Dialog コンテキストの管理」(P.3-20)
- 「ユーティリティ」(P.3-22)

要求行または応答行での操作

- 「`getRequestLine`」
- 「`getRequestUriParameter`」
- 「`setRequestUri`」
- 「`getResponseLine`」
- 「`setResponseCode`」

`getRequestLine`

`getRequestLine()` returns the method, request-uri, and version

このメソッドでは、次の 3 つの値が返されます。

- メソッド名
- 要求 URI
- プロトコルバージョン

この API が応答で起動されると、実行時エラーがトリガーされます。

例：メソッド、要求 URI、プロトコルバージョンのローカル変数を設定します。

スクリプト

```
M = {}

function M.outbound_INVITE(msg)
    local method, ruri, ver = msg:getRequestLine()
end

return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
```

出力/結果

```
Local variables method, ruri, and version are set accordingly:
method == "INVITE"
ruri == "sip:1234@10.10.10.1"
version == "SIP/2.0"
```

getRequestUriParameter

```
getRequestUriParameter(parameter-name) returns the URI parameter-value
```

この関数では、要求 URI パラメータの文字列名が指定された状態で、指定された URI パラメータの値が返されます。

- URI パラメータが「tag=value」の形式の場合、等記号の右側にある値が返されます。
- URI パラメータが値のないタグの場合、値は空文字列（例：""）で返されます。
- URI パラメータが最初のヘッダー値にない場合、nil が返されます。
- このメソッドが応答メッセージで起動された場合、nil が返されます。

例：要求 URI で、ローカル変数をユーザ パラメータの値に設定します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
    local userparam = msg:getRequestUriParameter("user")
end
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1;user=phone SIP/2.0
```

出力/結果

ローカル変数 userparam が、文字列「phone」に設定されます。

setRequestUri

```
setRequestUri(uri)
```

このメソッドでは、要求行の要求 URI が設定されます。この API が応答で起動されると、実行時エラーがトリガーされます。

例：要求 URI を tel:1234 に設定します。

スクリプト

```
M = {}  
function M.outbound_INVITE(msg)  
    msg:setRequestUri("tel:1234")  
end  
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
```

出力/結果

```
INVITE tel:1234 SIP/2.0
```

getResponseLine

```
getResponseLine() returns version, status-code, reason-phrase
```

このメソッドでは、プロトコルバージョン（文字列）、ステータスコード（整数）、および理由フレーズ（文字列）が返されます。この API が要求で起動されると、実行時エラーがトリガーされます。

例：ローカル変数を、プロトコルバージョン、ステータスコード、および理由フレーズの値に設定します。

スクリプト

```
M = {}  
function M.outbound_200_INVITE(msg)  
    local version, status, reason = msg:getResponseLine()  
end  
return M
```

メッセージ

```
SIP/2.0 200 Ok  
.  
.  
CSeq: 102 INVITE
```

出力/結果

```
Local variables method, ruri, and version are set accordingly:  
version == "SIP/2.0"  
status == 200  
reason == "Ok"
```

setResponseCode

```
setResponseCode(status-code, reason-phrase)
```

このメソッドでは、ステータスコードおよび理由フレーズが設定されます。いずれの値にも、nil を設定できます。値に nil が設定された場合、メッセージに保存されている既存値が使用されます。この API が要求で起動されると、実行時エラーがトリガーされます。

例：発信の 404 応答を 604 応答に変更します。

スクリプト

```
M = {}
function M.outbound_404_INVITE(msg)
  msg:setResponseCode(604, "Does Not Exist Anywhere")
end
return M
```

メッセージ

```
SIP/2.0 404 Not Found
.
.
CSeq: 102 INVITE
```

出力/結果

```
SIP/2.0 604 Does Not Exist Anywhere
.
.
CSeq: 102 INVITE
```

ヘッダーの操作

- 「allowHeader」
- 「getHeader」
- 「getHeaderValues」
- 「getHeaderValueParameter」
- 「getHeaderUriParameter」
- 「addHeader」
- 「addHeaderValueParameter」
- 「addHeaderUriParameter」
- 「modifyHeader」
- 「applyNumberMask」
- 「convertDiversionToHI」
- 「convertHIToDiversion」
- 「removeHeader」
- 「removeHeaderValue」

allowHeader

allowHeaders is a Lua table specified by the script writer

認識しないヘッダーに関連する Cisco Unified CM のデフォルト動作では、このようなヘッダーは無視されます。一部の場合、このようなヘッダーは正規化中に使用されるか、または、ヘッダーが透過的に渡される必要があります。**allowHeaders** テーブルでヘッダー名を使用することによって、スクリプトにより、Cisco Unified CM でヘッダーを認識できるようになり、正規化の場合はスクリプトでローカルに使用されるか、または透過的に渡されます。

例：

allowHeaders によって、History-Info が指定されます。指定がない場合、スクリプトの処理前に、着信 History-Info ヘッダーはドロップされます。したがって、**convertHIToDiversion** では、役に立つ結果は生成されません。このスクリプトでは、x-pbx-id と呼ばれる仮想ヘッダーも許可されます。これは、**allowHeaders** がヘッダー名のテーブルであることを示しています。

スクリプト

```
M = {}
M.allowHeaders = {"History-Info", "x-pbx-id"}
function M.inbound_INVITE(msg)
    msg:convertHIToDiversion()
end
return M
```

メッセージ

```
INVITE sip:1001@10.10.10.1 SIP/2.0
.
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1
```

結果

```
INVITE sip:1001@10.10.10.1 SIP/2.0
.
Diversion: <sip:2401@10.10.10.2>;reason=unconditional;counter=1
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1
```

getHeader

getHeader(header-name) returns header-value or nil

このメソッドでは、ヘッダーの文字列名が指定された状態で、ヘッダーの値が返されます。同じ名前の複数ヘッダーについては、値が連結され、カンマで区切られます。

例：ローカル変数を Allow ヘッダーの値に設定します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
    local allow = msg:getHeader("Allow")
end
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
To: <sip:1234@10.10.10.1>;
.
Allow: UPDATE
Allow: Ack,Cancel,Bye,Invite
```

出力/結果

```
Local variable allow set to the string "UPDATE,Ack,Cancel,Bye,Invite"
```

getHeaderValues

getHeaderValues(header-name) returns a table of header values

この関数では、ヘッダーの文字列名が指定された状態で、値がインデックス化されたテーブルとして、ヘッダーのカンマ区切りの値が返されます。Lua の索引は 1 に基づいています。ヘッダー値 *n* がある場合、最初の値はインデックス 1 にあり、最後の値はインデックス *n* にあります。Lua の **ipairs** 機能では、テーブル全体で繰り返し使用できます。

例：History-Info ヘッダーの値が含まれるローカル テーブルを作成します。

スクリプト

```
M = {}
M.allowHeaders = {"History-Info"}
function M.inbound_INVITE(msg)
    local history_info = msg.getHeaderValues("History-Info")
end
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
To: <sip:1234@10.10.10.1>;
.
History-Info: <sip:UserB@hostB?Reason=sip;cause=408>;index=1
History-Info: <sip:UserC@hostC?Reason=sip;cause=302>;index=1.1
History-Info: <sip:UserD@hostD>;index=1.1.1
```

出力/結果

```
Local variable history_info is set to
{
  "<sip:UserB@hostB?Reason=sip;cause=408>;index=1",
  "<sip:UserC@hostC?Reason=sip;cause=302>;index=1.1",
  "<sip:UserD@hostD>;index=1.1.1"
}
```

In other words:

```
history_info[1] == "<sip:UserB@hostB?Reason=sip;cause=408>;index=1"
history_info[2] == "<sip:UserC@hostC?Reason=sip;cause=302>;index=1.1"
history_info[3] == "<sip:UserD@hostD>;index=1.1.1"
```

getHeaderValueParameter

`getHeaderValueParameter(header-name, parameter-name)` returns the parameter-value

このメソッドでは、ヘッダーおよびヘッダー パラメータの名前が指定された状態で、指定されたヘッダー パラメータの値が返されます。同じ名前の複数のヘッダーでは、最初のヘッダー値のみが評価されます。

- ヘッダー パラメータが「tag=value」の形式の場合、等記号の右側にある値が返されます。
- ヘッダー パラメータが値のないタグの場合、値は空文字列（例：""）で返されます。
- ヘッダー パラメータが最初のヘッダー値にない場合、nil が返されます。

例：ローカル変数を、tag という名前のヘッダー パラメータの値に設定します。

スクリプト

```
M = {}
function M.outbound_180_INVITE(msg)
    local totag = msg:getHeaderValueParameter("To", "tag")
end
return M
```

メッセージ

```
SIP/2.0 180 Ringing
.
To: <sip:1234@10.10.10.1>;tag=32355SIPpTag0114
```

出力/結果

```
Local variable totag is set to the string "32355SIPpTag0114"
```

getHeaderUriParameter

`getHeaderUriParameter(header-name, parameter-name)` returns the URI parameter-value

このメソッドでは、ヘッダーの文字列名が指定された状態で、指定された URI パラメータの値が返されます。同じ名前の複数のヘッダーでは、最初のヘッダー値のみが評価されます。

- URI パラメータが「tag=value」の形式の場合、等記号の右側にある値が返されます。
- URI パラメータが値のないタグの場合、値は空文字列（例：""）で返されます。
- URI パラメータが最初のヘッダー値にない場合、nil が返されます。

例：ローカル変数を、user という名前の uri パラメータの値に設定します。

スクリプト

```
M = {}
function M.outbound_180_INVITE(msg)
    local userparam = msg:getHeaderUriParameter("To", "user")
end
return M
```

メッセージ

```
SIP/2.0 180 Ringing
.
To: <sip:1234@10.10.10.1;user=phone>;tag=32355SIPpTag0114
```

出力/結果

```
Local variable userparam is set to the string "phone"
```

addHeader

```
addHeader(header-name, header-value)
```

このメソッドでは、ヘッダーおよび値の文字列名が指定された状態で、ヘッダー値のリストの末尾に値が付加されます。

例：Allow ヘッダーに INFO を追加します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
    msg:addHeader("Allow", "INFO")
end
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
To: <sip:1234@10.10.10.1>;
Allow: Ack,Cancel,Bye,Invite
```

出力/結果

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
To: <sip:1234@10.10.10.1>;
Allow: Ack,Cancel,Bye,Invite,INFO
```

addHeaderValueParameter

```
addHeaderValueParameter(header-name, parameter-name [,parameter-value])
```

このメソッドでは、ヘッダー名、パラメータ名、およびオプションパラメータ値が指定された状態で、SIPメッセージで指定されたヘッダーが検索され、指定されたヘッダーパラメータおよびオプションパラメータ値がヘッダーに追加されます。SIPメッセージで、元のヘッダーが変更されたヘッダー値に置き換えられます。指定されたヘッダーに複数のヘッダー値がある場合、最初のヘッダーのみが変更されます。SIPメッセージに指定されたヘッダーのインスタンスがない場合、アクションは実行されません。

例 : color=blue ヘッダー パラメータを発信 Contact ヘッダーに追加します。

スクリプト

```
M = {}  
function M.outbound_INVITE(msg)  
    msg:addHeaderValueParameter("Contact", "color", "blue")  
end  
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
Contact: <sip:1234@10.10.10.2>
```

出力/結果

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
Contact: <sip:1234@10.10.10.12>;color=blue
```

addHeaderUriParameter

```
addHeaderUriParameter(header-name, parameter-name [,parameter-value])
```

このメソッドでは、ヘッダー名、パラメータ名、およびオプション パラメータ値が指定された状態で、SIP メッセージで指定されたヘッダーが検索され、指定されたヘッダー URI パラメータおよびオプション URI パラメータ値が、含まれている URI に追加されます。SIP メッセージで、元のヘッダーが変更されたヘッダー値に置き換えられます。指定されたヘッダーに複数のヘッダー値がある場合、最初のヘッダーのみが変更されます。SIP メッセージに指定されたヘッダーのインスタンスがない場合、または、指定されたヘッダーの最初のインスタンス内に有効な URI が見つからない場合、アクションは実行されません。

例 : P-Asserted-Identity URI に、user=phone パラメータを追加します。

スクリプト

```
M = {}  
function M.inbound_INVITE(msg)  
    msg:addHeaderUriParameter("P-Asserted-Identity", "user", "phone")  
end  
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
P-Asserted-Identity: <sip:1234@10.10.10.1>
```

出力/結果

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
P-Asserted-Identity: <sip:1234@10.10.10.1;user=phone>
```

modifyHeader

```
modifyHeader(header-name, header-value)
```

このメソッドでは、ヘッダーおよび値の文字列名が指定された状態で、既存のヘッダー値が指定された値で上書きされます。これは、`removeHeader` が起動された後で `addHeader` が起動される動作に似ています。

例：発信 INVITE メッセージの P-Asserted-Identity ヘッダーから、表示名を削除します。

スクリプト

```
M = {}  
function M.outbound_INVITE(msg)  
  -- Remove the display name from the PAI header  
  local pai = msg.getHeader("P-Asserted-Identity")  
  local uri = string.match(pai, "<.+>")  
  msg.modifyHeader("P-Asserted-Identity", uri)  
end  
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
P-Asserted-Identity: "1234" <1234@10.10.10.1>
```

出力/結果

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
P-Asserted-Identity: <1234@10.10.10.1>
```

applyNumberMask

```
applyNumberMask(header-name, mask)
```

このメソッドでは、ヘッダー名および番号マスクが指定された状態で、SIP メッセージで指定されたヘッダーが検索され、指定された番号マスクが、ヘッダー内に含まれている URI に追加されます。URI がヘッダー値から正常に解析された場合、解析された URI のユーザ部分に番号マスクが適用されます。SIP メッセージで、元のヘッダーが変更されたヘッダー値に置き換えられます。指定されたヘッダーに複数のインスタンスがある場合、ヘッダーの最初のインスタンスのみが変更されます。

次の条件のいずれかが満たされない場合、アクションは実行されません。

1. 指定されたヘッダーのインスタンスが、SIP メッセージにない場合
2. 指定されたヘッダーの最初のインスタンス内に、有効な URI がない場合
3. 指定されたマスク パラメータが空の文字列の場合

番号マスクの適用

マスク パラメータによって、ヘッダー URI のユーザ部に適用される変換が定義されます。マスク パラメータでは、大文字の X でワイルドカード文字が指定されます。たとえば、マスク「+1888XXXXXXX」が指定されている場合、マスクの挿入文字として「X」が使用されます。このマスクが例のユーザ部「4441234」に適用される場合、結果の文字列は「+18884441234」になります。

マスクされるユーザ部に見つかった文字数が、マスクのワイルドカード文字数よりも少ない場合、最も左にあるワイルドカード文字は「X」で残されます。前述のマスクを例のユーザ部「1234」に適用すると、結果の文字列は「+1888XXX1234」になります。マスクされるユーザ部に見つかった文字数が、マスクのワイルドカード文字数よりも多い場合、ユーザ部の最も左にある文字が切り捨てられます。たとえば、マスク「+1888XXXX」がユーザ部「4441234」に適用される場合、結果の文字列は「+18881234」になります。

例：着信 INVITE メッセージの P-Asserted-Identity ヘッダーで、番号マスクに番号を適用します。

スクリプト

```
M = {}  
function M.inbound_INVITE(msg)  
    msg:applyNumberMask("P-Asserted-Identity", "+1919476XXXX")  
end  
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
P-Asserted-Identity: <sip:1234@10.10.10.1>
```

出力/結果

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
P-Asserted-Identity: <sip:+19194761234@10.10.10.1>
```

convertDiversionToHI

```
convertDiversionToHI()
```

Cisco Unified CM では、リダイレクト情報を処理するための Diversion ヘッダーの送信がサポートされます。たとえば、Cisco Unified CM によってコールが転送される場合について考えます。このようなインスタンスでは、リダイレクト番号の送信に Cisco Unified CM で使用される Diversion ヘッダーが、SIP メッセージに含まれる場合があります。ただし、多くの SIP サーバでは、この目的で、Diversion ヘッダーの代わりに History-Info ヘッダーが使用されます。この API を使用して、Diversion ヘッダーが History-Info ヘッダーに変換されます。



(注)

一部の実装では、この API の代わりにまたはこの API に加えて、raw ヘッダー API (例: getHeader、addHeader、modifyHeader、removeHeader) の使用が必要です。

例：発信 INVITE メッセージで、Diversion ヘッダーを History-Info ヘッダーに変換します。次に、Diversion ヘッダーを削除します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
  if msg.getHeader("Diversion")
    then
      msg:convertDiversionToHI()
      msg.removeHeader("Diversion")
    end
  end
end
return M
```

メッセージ

```
INVITE sip:2400@10.10.10.2 SIP/2.0
.
Diversion: <sip:1002@10.10.10.1>;reason=unconditional;privacy=off;screen=yes
```

出力/結果

```
INVITE sip:2400@10.10.10.2 SIP/2.0
.
History-Info: <sip:1002@10.10.10.1?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:2400@10.10.10.2>;index=1.1
```

convertHIToDiversion

```
convertHIToDiversion()
```

Cisco Unified CM では、リダイレクト情報を処理するための Diversion ヘッダーの受信がサポートされます。たとえば、Cisco Unified CM に到達する前にコールが転送される場合について考えます。このような場合、リダイレクト番号の確立に Cisco Unified CM で使用される Diversion ヘッダーが、SIP メッセージに含まれる場合があります。ただし、多くの SIP サーバでは、この目的で、Diversion ヘッダーの代わりに History-Info ヘッダーが使用されます。



(注)

一部の実装では、この API の代わりにまたはこの API に加えて、raw ヘッダー API (例: getHeader、addHeader、modifyHeader、removeHeader) の使用が必要な場合があります。

例

allowHeaders では、「History-Info」を指定する必要があります。指定がない場合、スクリプトの処理前に、着信 History-Info ヘッダーはドロップされます。したがって、convertHIToDiversion の呼び出しによっては、役に立つ結果は生成されません。

スクリプト

```
M = {}
M.allowHeaders = {"History-Info"}
function M.inbound_INVITE(msg)
  msg:convertHIToDiversion()
end
return M
```

メッセージ

```
INVITE sip:1001@10.10.10.1 SIP/2.0
.
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1
```

結果

```
INVITE sip:1001@10.10.10.1 SIP/2.0
.
Diversion: <sip:2401@10.10.10.2>;reason=unconditional;counter=1
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1
```

removeHeader

```
removeHeader(header-name)
```

このメソッドでは、ヘッダーの文字列名が指定された状態で、メッセージからヘッダーが削除されます。

例：発信 INVITE メッセージから、Cisco-Guid ヘッダーを削除します。

```
M = {}
function M.outbound_INVITE(msg)
    msg.removeHeader("Cisco-Guid")
end
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: "1234" <1234@10.10.10.1>
Cisco-Guid: 1234-4567-1234
Session-Expires: 1800
```

出力/結果

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: "1234" <1234@10.10.10.1>
Session-Expires: 1800
.
```

removeHeaderValue

```
removeHeaderValue(header-name, header-value)
```

このメソッドでは、ヘッダーの文字列名およびヘッダー値が指定された状態で、メッセージからヘッダー値が削除されます。値にヘッダー値のみがあった場合、ヘッダーそのものが削除されます。

例：発信 INVITE メッセージのサポート ヘッダーから、X-cisco-srtp-fallback を削除します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
    msg.removeHeaderValue("Supported", "X-cisco-srtp-fallback")
end
```

```
end
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: "1234" 1234@10.10.10.1
Supported: timer, replaces, X-cisco-srtp-fallback
Session-Expires: 1800
```

出力/結果

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: "1234" 1234@10.10.10.1
Supported: timer, replaces
Session-Expires: 1800
```

SDP の操作

- [「getSdp」](#)
- [「setSdp」](#)
- [「removeUnreliableSdp」](#)

getSdp

getSdp() returns string

このメソッドでは、Lua 文字列として SDP が返されます。メッセージに SDP が含まれない場合、nil が返されます。

例：SDP（SDP がない場合は nil）が含まれるローカル変数を確立します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
    local sdp = msg:getSdp()
end
return M
```

メッセージ

```
INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
Via: SIP/2.0/TCP 172.18.197.88:5080;branch=z9hG4bK4bae847b2
From: <sip:579@172.18.197.88>;tag=9f12addc-04fc-4f0f-825b-3cd7d6dd8813-25125665
To: <sip:901@rawlings.cisco.com>
Date: Thu, 28 Jan 2010 01:09:39 GMT
Call-ID: cd80d100-b601e3d3-15-58c512ac@172.18.197.88
.
.
Max-Forwards: 69
Content-Type: application/sdp
Content-Length: 214

v=0
```

```

o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24580 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15

```

出力/結果

変数 `sdp` が次の文字列に設定されます。この文字列には、組み込みの復帰改行文字および改行文字（例：「`\r\n`」）が含まれていることに、注意してください。

```

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24580 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15

```

setSdp

```
setSdp(sdp)
```

このメソッドでは、SIPメッセージで指定された文字列を SDP Content 本体として保存します。SIPメッセージの Content-Length ヘッダーは、文字列の長さに自動的に更新されます。

例：発信 SDP から a=ptime の行を削除します。

スクリプト

```

M = {}
function M.outbound_INVITE(msg)
    local sdp = msg:getSdp()
    if sdp
    then
        -- remove all ptime lines
        sdp = sdp:gsub("a=ptime:%d*\r\n", "")
        -- store the updated sdp in the message object
        msg:setSdp(sdp)
    end
end
return M

```

メッセージ

```

INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
Via: SIP/2.0/TCP 172.18.197.88:5080;branch=z9hG4bK4bae847b2
From: <sip:579@172.18.197.88>;tag=9f12addc-04fc-4f0f-825b-3cd7d6dd8813-25125665
To: <sip:901@rawlings.cisco.com>
Date: Thu, 28 Jan 2010 01:09:39 GMT
Call-ID: cd80d100-b601e3d3-15-58c512ac@172.18.197.88
.
.

```

```

Max-Forwards: 69
Content-Type: application/sdp
Content-Length: 214

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24580 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15

```

出力/結果

前述のスキプトの実行後の最後の SIP メッセージは、次のようになります。

```

INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
Via: SIP/2.0/TCP 172.18.197.88:5080;branch=z9hG4bK4bae847b2
From: <sip:579@172.18.197.88>;tag=9f12addc-04fc-4f0f-825b-3cd7d6dd8813-25125665
To: <sip:901@rawlings.cisco.com>
Date: Thu, 28 Jan 2010 01:09:39 GMT
Call-ID: cd80d100-b601e3d3-15-58c512ac@172.18.197.88
.
.
Max-Forwards: 69

Content-Type: application/sdp
Content-Length: 214

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24580 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15

```

removeUnreliableSdp

```
removeUnreliableSdp()
```

このメソッドでは、18X メッセージから SDP が削除されます。SCDP を削除する前に、メッセージでは、PRACK が必要ではないことが確認されます。



(注)

Content-Length ヘッダーへの調整は、この API によって SDP が実際に削除されるときに、自動的に行われます。メッセージに他の Content 本体がない場合、Content-Type ヘッダーは自動的に削除されます。

例：着信 180 メッセージから SDP を削除します。

スクリプト

```
M = {}
```

```
function M.inbound_180_INVITE(msg)
    msg:removeUnreliableSdp()
end
return M
```

メッセージ

```
SIP/2.0 180 Ringing
.
Content-Type: application/sdp
Content-Length: 214

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24580 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

出力/結果

前述のスクリプトの実行後の最後の SIP メッセージは、次のようになります。

```
SIP/2.0 180 Ringing
.
Content-Length: 0
```

Content 本体の操作

- [「getContentBody」](#)
- [「addContentBody」](#)
- [「removeContentBody」](#)

getContentBody

`getContentBody(content-type)` returns the content-body, content-disposition, content-encoding, and content-language of the specified content-type

この関数では、`content-type` の文字列名が指定された状態で、`content-body`、`content-disposition`、`content-encoding`、および `content-language` が返されます。`content-disposition`、`content-encoding`、および `content-language` はオプションで、メッセージで指定されていない場合があります。メッセージで特定のヘッダーが指定されていない場合、その値には `nil` が返されます。

メッセージに `content-body` がない場合、返されるすべての項目に `nil` 値が含まれます。

例：発信 INFO メッセージの Content 情報が含まれるローカル変数を確立します。

スクリプト

```
M = {}
function M.inbound_INFO(msg)
    local b = msg:getContentBody("application/vnd.nortelnetworks.digits")
```

```
end
return M
```

発信メッセージ

```
INFO sip: 1000@10.10.10.1 SIP/2.0
Via: SIP/2.0/UDP 10.10.10.57:5060
From: <sip:1234@10.10.10.57>;tag=d3f423d
To: <sip:1000@10.10.10.1>;tag=8942
Call-ID: 312352@10.10.10.57
Cseq: 5 INFO
Content-Type: application/vnd.nortelnetworks.digits
Content-Length: 72
```

```
p=Digit-Collection
y=Digits
s=success
u=12345678
i=87654321
d=4
```

出力/結果

Local variable b is set to the string:

```
p=Digit-Collection
y=Digits
s=success
u=12345678
i=87654321
d=4
```

addContentBody

```
addContentBody(content-type, content-body [,content-disposition [,content-encoding
[,content-language]])
```

この API では、`content-type` および `content-body` が指定された状態で、メッセージに Content 本体が追加され、`content-length` ヘッダーに必要な変更が行われます。スクリプトでは、オプションで、ディスポジション、エンコード、および言語が提供される場合があります。

例：発信 UPDATE メッセージに、プレーン テキストの Content 本体を追加します。

スクリプト

```
M = {}
function M.outbound_UPDATE(msg)
    msg:addContentBody("text/plain", "d=5")
end
return M
```

正規化前のメッセージ

```
UPDATE sip: 1000@10.10.10.1 SIP/2.0
.
Content-Length: 0
```

正規化後のメッセージ

```
UPDATE sip: 1000@10.10.10.1 SIP/2.0
.
```

```
Content-Length: 3
Content-Type: text/plain
d=5
```

removeContentBody

```
removeContentBody(content-type)
```

この API では、`content-type` が指定された状態で、関連付けられている Content 本体がメッセージから削除され、`content-length` ヘッダーに必要な変更が行われます。

例：発信 UPDATE メッセージからテキスト/プレーン Content 本体を削除します。

スクリプト

```
M = {}
function M.outbound_UPDATE(msg)
    msg.removeContentBody("text/plain")
end
return M
```

正規化前のメッセージ

```
UPDATE sip: 1000@10.10.10.1 SIP/2.0
.
Content-Length: 3
Content-Type: text/plain

d=5
```

正規化後のメッセージ

```
UPDATE sip: 1000@10.10.10.1 SIP/2.0
.
Content-Length: 0
```

メッセージのブロック

- 「[block](#)」

block

```
block()
```

このメソッドでは、着信または発信の信頼できない 18X メッセージをブロックできます。着信のブロックでは、メッセージを受信しなかったかのように CUCM が動作します。発信のブロックでは、CUCM によってネットワークにメッセージが送信されません。



(注) PRACK が有効で、メッセージが実際は信頼可能な 18X の場合、CUCM では `block()` への呼び出しは事実上無視されます。メッセージはブロックされず、エラー SDI トレースが生成されます。

例：発信 183 メッセージをブロックします。

スクリプト

```
M = {}
function M.outbound_183_INVITE(msg)
    msg:block()
end
return M
```

メッセージ

```
SIP/2.0 183 Progress
Via: SIP/2.0/TCP 172.18.199.186:5060;branch=z9hG4bK5607aa91b
From: <sip:2220@172.18.199.186>;tag=7cae99f9-2f13-4a4d-b688-199664cc38a4-32571707
To: <sip:2221@172.18.199.100>;tag=3affe34f-6434-4295-9a4b-clfe2b0e03b6-21456114
.
.
```

出力/結果

The 183 message is not sent to the network.

透過性

- [「getPassThrough」](#)

getPassThrough

getPassThrough() returns a per message pass through object

このメソッドでは、SIP パス スルー オブジェクトが返されます。これは、着信メッセージに対して起動できます。パス スルー オブジェクトに追加される情報は、他のコール レッグで生成された発信メッセージに自動的に結合されます。この情報の自動結合は、(必要に応じて) 発信の正規化の前に発生します。パス スルー オブジェクトの使用および例については、[「SIP パススルー API」](#) を参照してください。

Per-Dialog コンテキストの管理

- [「getContext」](#)

getContext

getContext() returns a per call context

このメソッドでは、コールごとのコンテキストが返されます。コンテキストは Lua テーブルです。このスクリプト作成者は、ダイアログの動作中に使用される多様なメッセージ ハンドラで、コンテキストにデータを保存し、コンテキストからデータを取得することができます。スクリプト作成者は、ダイアログの最後でコンテキストの解放を考える必要はありません。コンテキストは、ダイアログの終了時に、Cisco Unified CM によって自動的に解放されます。

例：

このスクリプトでは、コンテキストが使用され、INVITE で受信した History-Info ヘッダーが保存されます。INVITE 送信の応答時に、保存されたヘッダーが取得され、発信応答に追加されます。「clear」パラメータが使用され、後続の reINVITE 応答へのこれらのヘッダーの追加が防止されます。

スクリプト

```
M = {}
M.allowHeaders = {"History-Info"}
function M.inbound_INVITE(msg)
    local history = msg.getHeader("History-Info")
    if history
    then
        msg:convertHIToDiversion()
        local context = msg.getContext()
        if context
        then
            context["History-Info"] = history
        end
    end
end

local function includeHistoryInfo(msg, clear)
    local context = msg.getContext()
    if context
    then
        local history = context["History-Info"]
        if history
        then
            msg.addHeader("History-Info", history)

            if clear
            then
                context["History-Info"] = nil
            end
        end
    end
end

function M.outbound_18X_INVITE(msg)
    includeHistoryInfo(msg)
end

function M.outbound_200_INVITE(msg)
    includeHistoryInfo(msg, "clear")
end
return M
```

正規化前のメッセージ

```
INVITE sip:1001@10.10.10.1 SIP/2.0
.
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1

SIP/2.0 180 Ringing
.
Content-Length: 0

SIP/2.0 200 OK
.
Content-Type: application/sdp
```

正規化後のメッセージ

```

INVITE sip:1001@10.10.10.1 SIP/2.0
.
Diversion: <sip:2401@10.10.10.2>;reason=unconditional;counter=1
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1
History-Info: <sip:1001@10.10.10.1>;index=1.1

SIP/2.0 180 Ringing
.
Content-Length: 0
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1,
<sip:1001@10.10.10.1>;index=1.1

SIP/2.0 200 OK
.
Content-Type: application/sdp
History-Info: <sip:2401@10.10.10.2?Reason=sip;cause=302;text="unconditional">;index=1,
<sip:1001@10.10.10.1>;index=1.1
.

```

ユーティリティ

- 「[isInitialInviteRequest](#)」
- 「[isReInviteRequest](#)」
- 「[getUri](#)」

isInitialInviteRequest

`isInitialInviteRequest()` returns true or false

このメソッドでは、メッセージが要求された場合、メソッドが INVITE の場合、および、To ヘッダーにタグパラメータがない場合に、真が返されます。これ以外の場合は、偽が返されます。

例:

発信 INVITE が初期 INVITE かどうかに基づいて、ローカル変数を設定します。この例では、INVITE は初期 INVITE で、したがって、この場合の INVITE については、値は真になります。

スクリプト

```

M = {}
function M.outbound_INVITE(msg)
    local isInitialInvite = msg.isInitialInviteRequest()
end
return M

```

メッセージ

```

INVITE sip:1234@10.10.10.1 SIP/2.0
.
To: <sip:1234@10.10.10.1>

```

出力/結果

```
Local variable isInitialInvite set to 'true'
```

例 :

発信 INVITE が初期 INVITE かどうかに基づいて、ローカル変数を設定します。この例では、INVITE は初期 INVITE ではなく、したがって、この場合の INVITE については、値は偽になります。

スクリプト

```
M = {}  
function M.outbound_INVITE(msg)  
    local isInitialInvite = msg:isInitialInviteRequest()  
end  
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
To: <sip:1234@10.10.10.1>;tag=1234
```

出力/結果

```
Local variable isInitialInvite set to 'false'.
```

isReInviteRequest

```
isReInviteRequest() returns true or false
```

このメソッドでは、メッセージが要求された場合、メソッドが INVITE の場合、および、To ヘッダーにタグパラメータが1つある場合に、真が返されます。これ以外の場合は、偽が返されます。

例 :

発信 INVITE が reINVITE かどうかに基づいて、ローカル変数を設定します。この例では、INVITE は reINVITE で、したがって、この場合の INVITE については、値は真になります。

スクリプト

```
M = {}  
function M.outbound_INVITE(msg)  
    local isReInvite = msg:isReInviteRequest()  
end  
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
To: <sip:1234@10.10.10.1>;tag=112233445566  
.
```

出力/結果

```
Local variable isReInvite set to 'true'.
```

例 :

発信 INVITE が reINVITE かどうかに基づいて、ローカル変数を設定します。この例では、INVITE は reINVITE ではなく、したがって、この場合の INVITE については、値は偽になります。

スクリプト

```
M = {}
function M.outbound_ANY(msg)
    local isReInvite = msg:isReInviteRequest()
end
return M
```

メッセージ

```
CANCEL sip:1234@10.10.10.1 SIP/2.0
.
```

出力/結果

ローカル変数 `isReInvite` が「false」に設定されます。

getUri

`getUri(header-name)` returns a string or nil

このメソッドでは、ヘッダー名が指定された状態で、SIP メッセージで指定されたヘッダーが検索され、そのヘッダーからの URI の取得が試行されます。指定されたヘッダーに複数のインスタンスがある場合、ヘッダーの最初のインスタンスの URI が返されます。SIP メッセージに指定されたヘッダーのインスタンスがない場合、または、指定されたヘッダーの最初のインスタンス内に有効な URI が見つからない場合、`Lua nil` が返されます。

例：P-Asserted-Identity ヘッダーで、ローカル変数を URI に設定します。

スクリプト

```
M = {}
function M.inbound_INVITE(msg)
    local uri = msg:getUri("P-Asserted-Identity")
end
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: <sip:1234@10.10.10.1>
.
```

出力/結果

Local variable `uri` is set to "sip:1234@10.10.10.1"



CHAPTER 4

SDP API

この章では、Session Description Protocol (SDP; セッション記述プロトコル) のコンテンツ本文に関連した API について説明します。SDP は、SIP Message オブジェクトとは違って、基本文字列として扱われます。このため、スクリプト作成者は、Lua の文字列ライブラリの機能を利用することが可能です。Lua 文字列ライブラリの関数に加えて、シスコからも SDP 操作を支援するための API をいくつかご用意しています。Cisco API は文字列ライブラリに直接追加されますが、

スクリプトで SDP を操作するには、まず Lua SIP Message オブジェクトで提供される `getSdp()` API を使用して Lua SIP Message オブジェクトから SDP コンテンツ本文を取得する必要があります。その後は、スクリプト内で Cisco API に含まれる文字列ライブラリを使用して、SDP を操作できます。修正を終えたら、SIP Message オブジェクトに用意されている `setSdp(sdp)` API を使用して、SDP を SIP Message オブジェクトに書き戻します。これらの API の詳細については、次の「[SIP メッセージの API](#)」を参照してください。

```
-- Get the SDP content body from the SIP message
local sdp = msg:getSdp()
-- modification of the SDP happens at this point
-- Update the SDP associated with the SIP message
msg:setSdp(sdp)
```

シスコからは、SDP を操作するという目的で、個々の行およびメディア記述全体の取得、修正、追加、挿入、削除を行う API を用意しています。この章の以降の部分では、これらの Cisco API について説明します。

SDP の任意の行を取得して操作するには、行を識別する手段が必要です。スクリプト作成者は、無効な SDP に影響できなければなりません。したがって、正規化のために SDP の構造を知っておくことが最低限必要です。無効な SDP に影響することは、不可能ではないにしても難しいです。有効な SDP を操作することも可能です。これらの目標を念頭に置いて、文字列ベースのアプローチが保証されています。

SDP を操作するための Lua API は、大きく分けると 2 つのカテゴリに分類されます。

- 「[行レベルの API](#)」: この一連の API は、SDP 内の行を取得して操作できるようになっています。
- 「[メディア記述レベルの API](#)」: この一連の API は、特定のメディア記述を操作するためのものです。

セッション レベルの行の操作には、行レベルの一連の API を使用し、メディア記述の操作には、メディア記述レベルの一連の API を使用します。



(注)

行を操作するために設計された一連の API の操作対象は、セッション レベルの行のみに限定されるわけではありません。

行レベルの API

Cisco API より

次の行レベル Cisco API が使用可能です。

- 「[getLine](#)」 (start-of-line, line-contains)
- 「[modifyLine](#)」 (start-of-line, line-contains, new-line)
- 「[addline](#)」 (new-line)
- 「[insertLineAfter](#)」 (start-of-line, line-contains, new-line)
- 「[insertLineBefore](#)」 (start-of-line, line-contains, new-line)
- 「[removeLine](#)」 (start-of-line, line-contains)

パラメータの説明

パラメータ	説明
sdp	sdp パラメータは、SDP の入った文字列です。制御文字（「\r」、「\n」など）もすべて含みます。API は、sdp パラメータについて SDP 構造を一切強制しません。単なる文字列として扱います。スクリプト作成者は、SDP の一部分（たとえば、音声メディア記述など）を操作するのにこの API を使用できます。
start-of-line	start-of-line パラメータは、sdp 内の行の開始を探すのに使用される文字列です。一般的なスクリプトでは、たとえば、「o=」、「a=rtpmap:9」などが sdp 内のこの文字列で始まる最初の行の取得/修正/挿入/削除に使用できます。この文字列で始まる行が複数存在する場合は、最初の行だけが操作対象となります。 このパラメータは、<char>=<string> という形式になっていなければなりません。つまり、2 文字目が必ず等号でなくてはなりません。このパラメータがこの形式になっていない場合は、「nil」が返されます。
line-contains	line-contains は、「start-of-line」パラメータに指定した文字列で開始する行内で検索する、追加の一致条件として使用される文字列です。nil を指定した場合、 line-contains は一致条件として使用されません。その場合、 start-of-line パラメータだけが行の検索に使用されます。 line-contains が nil でなければ、 start-of-line と line-contains の両方が検索に使用されます。SDP 内に検索条件（両方）に一致する行が複数存在する場合は、最初の行だけが返されます。 このパラメータで指定するこの追加の一致条件が必要ない場合は、値として「nil」を引用符なしで渡す必要があります。
new-line	new-line は、最終的な sdp（返された sdp）に追加/挿入する文字列です。

上記 API は、これらのパラメータを文字列として扱います。このため、セッション レベルの行とメディア レベルの行の両方を操作できます。行がセッション レベルかメディア レベルかに応じて動作が変わることは一切ありません。条件に一致する行を検索する際、各行は \n（または \r\n）で終わるものとします。

パラメータ内の文字はいずれも Lua の「マジック キャラクタ」とは見なされません。すべての文字が、特別な意味は一切持たず、プレーン文字として扱われます。

getLine

`sdp:getLine(start-of-line, line-contains)` returns string

このメソッドは、`sdp` 内の **start-of-line** で開始して文字列 **line-contains** を持つ最初の行を返します。

例：SDP から `c` 行を取得して、ローカル変数に値として格納します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
    local sdp = msg:getSdp()
    if sdp
    then
        local ipv4_c_line = sdp:getLine("c=", "IP4")
    end
end
return M
```

メッセージ

```
INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
.
Content-Type: application/sdp

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24580 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

出力/結果

Local variable `ipv4_c_line` is set to `"c=IN IP4 172.18.197.88\r\n"`

modifyLine

`sdp:modifyLine(start-of-line, line-contains, new-line)`

このメソッドは、「`sdp`」内で **start-of-line** で開始する行を探します。**line-contains** パラメータが `nil` でない場合は、それもその行内に存在する必要があります。一致する行（行終了文字を含む）が、**new-line** パラメータで置き換えられます。得られた `sdp` が返されます。

一致する行が見つからない場合は、元の `sdp` が未変更のまま返されます。



(注)

SDP のルールに従うために、**new-line** には `\r\n`（または `\n`）で終了する文字列を指定することを推奨します。ユーティリティによって行終了文字が強制されることはありません。

例：次のコードは、a= 行の G.722 コーデックをドットのない G722 に変更します。

スクリプト

```
M = {}

function M.inbound_INVITE(msg)
    local sdp = msg:getSdp()

    if sdp
    then
        local g722_line = sdp:getLine("a=", "G.722")

        if g722_line
        then
            --Replace G.722 with G722. The dot is special and must be
            --escaped using % when using gsub.
            g722_line = g722_line:gsub("G%.722", "G722")
            sdp = sdp:modifyLine("a=", "G.722", g722_line)
            msg:setSdp(sdp)
        end
    end
end

return M
```

メッセージ

```
INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
.
Content-Type: application/sdp

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.22
s=SIP Call
c=IN IP4 172.18.197.29
t=0 0
m=audio 23588 RTP/AVP 9 0 8 18 101
a=rtpmap:9 G.722/8000
aptime:20
a=rtpmap:0 PCMU/8000
aptime:20
a=rtpmap:8 PCMA/8000
aptime:20
a=rtpmap:18 G729/8000
aptime:20
a=fmtp:18 annexb=no
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

出力/結果

```
INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
.
Content-Type: application/sdp

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.22
s=SIP Call
c=IN IP4 172.18.197.29
t=0 0
m=audio 23588 RTP/AVP 9 0 8 18 101
a=rtpmap:9 G722/8000
aptime:20
```

```

a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:8 PCMA/8000
a=ptime:20
a=rtpmap:18 G729/8000
a=ptime:20
a=fmtp:18 annexb=no
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15

```

addline

```
sdp:addLine(new-line)
```

このメソッドは、SDPの末尾に **new-line** を追加します。SDPのルールに従うために、**new-line** には `\r\n` (または `\n`) で終了する文字列を指定することを推奨します。このAPIによって行終了文字が強制されることはありません。

例：SDPの末尾に属性行を追加します。

スクリプト

```

M = {}

function M.outbound_INVITE(msg)
    local sdp = msg:getSdp()

    if sdp
    then
        sdp = sdp:addLine("a=some-attribute\r\n")

        msg:setSdp(sdp)
    end
end

return M

```

メッセージ

```

INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
.
Content-Type: application/sdp

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24690 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15

```

出力/結果

```

INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
.
Content-Type: application/sdp

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.88
s=SIP Call
c=IN IP4 172.18.197.88
t=0 0
m=audio 24690 RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=some-attribute

```

insertLineAfter

```
sdp:insertLineAfter(start-of-line, line-contains, new-line)
```

このメソッドは、sdp 内で **start-of-line** で開始し、なおかつ **line-contains** パラメータが指定されていればその行内に **line-contains** も持つ最初の行を探します。次に、見つかった行の後ろに **new-line** を挿入します。

例：SDP 内の「a=」および「G729」を持つ行の後ろに行を 1 つ挿入します。

スクリプト

```

M = {}

function M.outbound_INVITE(msg)
    local sdp = msg:getSdp()

    if sdp
    then
        sdp = sdp:insertLineAfter("a=", "G729", "a=ptime:30\r\n")
        msg:setSdp(sdp)
    end
end

return M

```

メッセージ

```

INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
.
Content-Type: application/sdp

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.22
s=SIP Call
c=IN IP4 172.18.197.29
t=0 0
m=audio 21702 RTP/AVP 18 101
a=rtpmap:18 G729/8000
a=fmtp:18 annexb=no
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15

```

出力/結果

```
INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
.
Content-Type: application/sdp

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.22
s=SIP Call
c=IN IP4 172.18.197.29
t=0 0
m=audio 21702 RTP/AVP 18 101
a=rtpmap:18 G729/8000^M
a=ptime:30
a=fmtp:18 annexb=no
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

insertLineBefore

```
sdp:insertLineBefore(start-of-line, line-contains, new-line)
```

このメソッドは、sdp 内で **start-of-line** で開始し、なおかつ **line-contains** パラメータが指定されていればその行内に **line-contains** も持つ最初の行を探します。次に、見つかった行の前に **new-line** を挿入します。

例：SDP 内の「s=」行の前に行を 1 つ挿入します。

スクリプト

```
M = {}

function M.inbound_ANY_INVITE(msg)
    local sdp = msg:getSdp()

    if sdp
    then
        sdp= sdp:insertLineBefore("s=", nil, "e=nobody@cisco.com\r\n")
        msg:setSdp(sdp)
    end
end

return M
```

メッセージ

```
SIP/2.0 200 OK
.
Content-Type: application/sdp

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.22
s=SIP Call
c=IN IP4 172.18.197.29
t=0 0
m=audio 17774 RTP/AVP 9 101
a=rtpmap:9 G722/8000
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

出力/結果

```
SIP/2.0 200 OK
.
Content-Type: application/sdp

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.22
e=nobody@cisco.com
s=SIP Call
c=IN IP4 172.18.197.29
t=0 0
m=audio 17774 RTP/AVP 9 101
a=rtpmap:9 G722/8000
aptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

removeLine

```
sdp:removeLine(start-of-line, line-contains)
```

このメソッドは、「sdp」内で「start-of-line」で開始し、なおかつ「line-contains」パラメータが指定されていればその行内に「line-contains」も持つ最初の行を探します。一致した行は、sdp から削除されます。

例：「a=rtpmap」と「G729」の両方を含む行を SDP から削除します。

スクリプト

```
M = {}

function M.inbound_INVITE(msg)
    local sdp = msg:getSdp()

    if sdp
    then
        sdp = sdp:removeLine("a=rtpmap:", "G729")
        msg:setSdp(sdp)
    end
end

return M
```

メッセージ

```
INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
.
Content-Type: application/sdp

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.22
s=SIP Call
c=IN IP4 172.18.197.29
t=0 0
m=audio 25328 RTP/AVP 9 0 8 18 101
a=rtpmap:9 G722/8000
aptime:20
a=rtpmap:0 PCMU/8000
aptime:20
a=rtpmap:8 PCMA/8000
aptime:20
```

```

a=rtpmap:18 G729/8000
a=ptime:20
a=fmtp:18 annexb=no
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15

```

出力/結果

```

INVITE sip:901@rawlings.cisco.com:5080 SIP/2.0
.
Content-Type: application/sdp

v=0
o=CiscoSystemsCCM-SIP 2000 1 IN IP4 172.18.197.22
s=SIP Call
c=IN IP4 172.18.197.29
t=0 0
m=audio 25328 RTP/AVP 9 0 8 18 101
a=rtpmap:9 G722/8000
a=ptime:20
a=rtpmap:0 PCMU/8000
a=ptime:20
a=rtpmap:8 PCMA/8000
a=ptime:20
a=ptime:20
a=fmtp:18 annexb=no
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15

```

メディア記述レベルの API

SDP メディアレベルでの SDP の取得と操作に使用される API は次のとおりです。

SDP の例

次の SDP の例では、セッション レベルの行は緑色で示し、メディア レベルの行を青色で示しています。

```

v=0\r\n
  o=CiscoSystemsCCM-SIP 2000 3 IN IP4 172.18.195.96\r\n
  s=SIP Call\r\n
  c=IN IP4 172.18.195.126\r\n
  t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
    a=rtpmap:9 G722/8000\r\n
    a=ptime:20\r\n
    a=rtpmap:18 G729/8000\r\n
    a=ptime:20\r\n
    a=fmtp:18 annexb=no\r\n
    a=rtpmap:101 telephone-event/8000\r\n
    a=fmtp:101 0-1\r\n
[2] m=video...

```

上の SDP の例では、メディア レベルのインデックスは青色になっています。メディア レベル [1] は、音声メディアの記述です。メディア レベル [2] は、ビデオメディアの記述です(全体は示していません)。

インデックスはすべて、1 から始まるインデックスです。したがって、最初のメディア記述はインデックス 1、2 つ目はインデックス 2 となります。以下、同様です。

メディア記述レベルでは、次の API が使用できます。

- 「`getMediaDescription(media-level)`」
- 「`getMediaDescription(media-contains)`」
- 「`modifyMediaDescription(media-level, media-description)`」
- 「`modifyMediaDescription(media-contains, media-description)`」
- 「`addMediaDescription(media-description)`」
- 「`insertMediaDescription(media-level, media-description)`」
- 「`removeMediaDescription(media-level)`」
- 「`removeMediaDescription(media-contains)`」

パラメータの説明

パラメータ	説明
sdp	sdp オブジェクトは、SDP の入った文字列です。制御文字（「 <code>\r</code> 」、「 <code>\n</code> 」など）もすべて含みます。
media-level	media-level パラメータは、SDP 内のメディア記述を参照するために使用されるインデックスです。インデックスは、1 から始まります。SDP 内の最初のメディア記述は、メディア レベル 1、2 つ目はメディア レベル 2 です。以下、同様です。
media-contains	media-contains パラメータは、m 行内で何か（例：メディア タイプ）を探すために使用される文字列です。 たとえば、audio、video、message などを使用して、メディア記述内の m 行が media-contains 値に完全に一致する最初のメディア記述を取得したり、修正したり、削除したりできます。
media-description	media-description パラメータは、複数の行と必要な埋め込み制御文字（「 <code>\r</code> 」および「 <code>\n</code> 」）を含む完全なメディア記述が入った文字列です。 最後の行は「 <code>\r\n</code> 」で終了させることを強く推奨します。これらの API が行う内容は純粋なテキスト操作で、テキストに対して強制される形式に関するルールは一切ありません。実際、実装によっては、各行の末尾に「 <code>\r</code> 」が含まれない場合もあります。これを純粋なテキストとして扱うことにより、いずれのケースも処理することができ、2 つのスタイル間での変換さえも簡単にできるという、必要性のある柔軟性が得られます（たとえば、「 <code>\r\n</code> 」を「 <code>\n</code> 」に変換したり、その逆を行ったりすることはごく普通のことです）。 次の文字列は、完全なメディア記述の一例です。 <code>"m=audio 18884 RTP/AVP 9\r\na=rtpmap:9 G722/8000\r\n"</code>

以降で、各メディア関連 API の詳細について説明します。



(注)

インデックス（つまり media-level）は、すべて 1 から始まるインデックスです。

getMediaDescription(media-level)

getMediaDescription(media-level) returns media-description

この API は、指定したメディア レベルの特定のメディア記述を取得するのに使用します。

例：最初のメディア記述を取得して、ローカル変数に値として格納します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
    local sdp = msg:getSdp()
    if sdp
    then
        local m1 = sdp:getMediaDescription(1)
    end
end
return M
```

メッセージ

```
v=0\r\n
o=CiscoSystemsCCM-SIP 2000 3 IN IP4 172.18.195.96\r\n
s=SIP Call\r\n
c=IN IP4 172.18.195.126\r\n
t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
a=ptime:20\r\n
a=rtpmap:18 G729/8000\r\n
a=ptime:20\r\n
a=fmtp:18 annexb=no\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
[2] m=video...
```

出力/結果

```
Local variable m1 is set to a string

m=audio 18884 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
a=ptime:20\r\n
a=rtpmap:18 G729/8000\r\n
a=ptime:20\r\n
a=fmtp:18 annexb=no\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
```

getMediaDescription(media-contains)

getMediaDescription(media-contains) returns media-description

この API は、m 行に media-contains と完全に一致する部分を持つ特定のメディア記述を取得するために使用します。



(注)

複数の m 行が一致する場合は、media-description と一致する最初のメディア記述だけが返されます。

例： m 行に「audio」というテキストを含む最初のメディア記述を取得して、ローカル変数に値として格納します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
  local sdp = msg:getSdp()
  if sdp
  then
    local audio = sdp:getMediaDescription("audio")
  end
end
return M
```

メッセージ

```
v=0\r\n
o=CiscoSystemsCCM-SIP 2000 3 IN IP4 172.18.195.96\r\n
s=SIP Call\r\n
c=IN IP4 172.18.195.126\r\n
t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
aptime:20\r\n
a=rtpmap:18 G729/8000\r\n
aptime:20\r\n
a=fmtp:18 annexb=no\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
[2] m=video...
```

出力/結果

```
Local variable audio is set to a string

m=audio 18884 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
aptime:20\r\n
a=rtpmap:18 G729/8000\r\n
aptime:20\r\n
a=fmtp:18 annexb=no\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
```

modifyMediaDescription(media-level, media-description)

```
modifyMediaDescription(media-level, media-description)
```

この API は、指定したメディア レベルの特定のメディア記述を修正するのに使用します。

例：「a=ptime:20\r\n」を含む業を音声メディア記述から削除します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
    local sdp = msg:getSdp()
    if sdp
    then
        local m1 = sdp:getMediaDescription(1)
        if m1
        then
            m1 = m1:gsub("a=ptime:20\r\n", "")
            sdp = sdp:modifyMediaDescription(1, m1)
            msg:setSdp(sdp)
        end
    end
end
return M
```

正規化前のメッセージ

```
v=0\r\n
    o=CiscoSystemsCCM-SIP 2000 3 IN IP4 172.18.195.96\r\n
    s=SIP Call\r\n
    c=IN IP4 172.18.195.126\r\n
    t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
    a=rtpmap:9 G722/8000\r\n
    a=ptime:20\r\n
    a=rtpmap:18 G729/8000\r\n
    a=ptime:20\r\n
    a=fmtp:18 annexb=no\r\n
    a=rtpmap:101 telephone-event/8000\r\n
    a=fmtp:101 0-1\r\n
[2] m=video...
```

正規化後のメッセージ

After normalization, the SDP would contain:

```
v=0\r\n
    o=CiscoSystemsCCM-SIP 2000 3 IN IP4 172.18.195.96\r\n
    s=SIP Call\r\n
    c=IN IP4 172.18.195.126\r\n
    t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
    a=rtpmap:9 G722/8000\r\n
    a=rtpmap:18 G729/8000\r\n
    a=fmtp:18 annexb=no\r\n
    a=rtpmap:101 telephone-event/8000\r\n
    a=fmtp:101 0-1\r\n
[2] m=video...
```

modifyMediaDescription(media-contains, media-description)

```
modifyMediaDescription(media-contains, media-description)
```

この API は、m 行に media-contains と完全に一致する部分を持つ特定のメディア記述を修正するために使用します。複数の m 行が一致する場合は、media-description と一致する最初のメディア記述だけが修正されます。

例：「a=ptime:20\r\n」を含む業を音声メディア記述から削除します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
  local sdp = msg:getSdp()
  if sdp
  then
    local audio = sdp:getMediaDescription("audio")
    if audio
    then
      audio = audio:gsub("a=ptime:20\r\n", "")
      sdp = sdp:modifyMediaDescription("audio", audio)
      msg:setSdp(sdp)
    end
  end
end
return M
```

正規化前のメッセージ

```
v=0\r\n
o=CiscoSystemsCCM-SIP 2000 3 IN IP4 172.18.195.96\r\n
s=SIP Call\r\n
c=IN IP4 172.18.195.126\r\n
t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
a=ptime:20\r\n
a=rtpmap:18 G729/8000\r\n
a=ptime:20\r\n
a=fmtp:18 annexb=no\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
[2] m=video...
```

正規化後のメッセージ

```
v=0\r\n
o=CiscoSystemsCCM-SIP 2000 3 IN IP4 172.18.195.96\r\n
s=SIP Call\r\n
c=IN IP4 172.18.195.126\r\n
t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
a=ptime:20\r\n
a=rtpmap:18 G729/8000\r\n
a=ptime:20\r\n
a=fmtp:18 annexb=no\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
[2] m=video...
```

addMediaDescription(media-description)

```
addMediaDescription(media-description)
```

この API は、1 つのメディア記述を最後のメディア記述として追加するのに使用します。

例：

このスクリプトは、着信 INVITE からビデオ メディア記述を削除して、保存します。このスクリプトでは、ビデオ メディア記述が 2 つ目のメディア記述であることを前提としています。対応する発信応答については、このスクリプトは、格納されているビデオ メディア記述を取得し、ポートを 0 に設定し、そのメディア記述を発信 SDP に追加します。

スクリプト

```
M = {}
function M.inbound_INVITE(msg)
    local sdp = msg:getSdp()
    if sdp
    then
        local video = sdp:getMediaDescription(2)
        if video
        then
            --remove the video media description
            sdp = sdp:removeMediaDescription(2)
            --store video media description
            local context = msg:getContext()
            context["video"] = video
            msg:setSdp(sdp)
        end
    end
end

function M.outbound_ANY_INVITE(msg)
    local sdp = msg:getSdp()
    --assume other side is expecting video before audio when there is
    --a video m-line in the SDP
    if sdp
    then
        local context = msg:getContext()
        local video = context["video"]
        if video
        then
            --set port to zero in SDP answer
            video = video:gsub("video %d* RTP", "video 0 RTP")
            sdp = sdp:addMediaDescription(video)
            msg:setSdp(sdp)
        end
    end
end
return M
```

正規化前のメッセージ

```
INVITE SDP . . .

v=0\r\n
o=- 2000 3 IN IP4 10.10.10.100\r\n
s=SIP Call\r\n
c=IN IP4 10.10.10.100\r\n
t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
```

```

a=ptime:20\r\n
a=rtpmap:18 G729/8000\r\n
a=ptime:20\r\n
a=fmtp:18 annexb=no\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
[2] m=video 32068 RTP/AVP 112 113 114\r\n
b=TIAS:4000000\r\n
a=rtpmap:112 H264/90000\r\n
a=fmtp:112
profile-level-id=4D8028;packetization-mode=1;max-mbps=243000;max-fs=8100\r\n
a=rtpmap:113 H264/90000\r\n
a=fmtp:113 profile-level-id=42400D;packetization-mode=1;max-mbps=11880;max-fs=396\r\n
a=rtpmap:114 H264/90000\r\n
a=fmtp:114 profile-level-id=42400D;packetization-mode=0;max-mbps=11880;max-fs=396\r\n
a=sendrecv\r\n
\r\n

200 Ok SDP . . .

v=0\r\n
o=CiscoSystemsCCM-SIP 2000 3 IN IP4 10.10.10.1\r\n
s=SIP Call\r\n
c=IN IP4 10.10.10.200\r\n
t=0 0\r\n
[1] m=audio 16007 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
a=ptime:20\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
\r\n

```

正規化後のメッセージ

```

INVITE SDP . . .

v=0\r\n
o=- 2000 3 IN IP4 10.10.10.100\r\n
s=SIP Call\r\n
c=IN IP4 10.10.10.100\r\n
t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
a=ptime:20\r\n
a=rtpmap:18 G729/8000\r\n
a=ptime:20\r\n
a=fmtp:18 annexb=no\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
\r\n

200 Ok SDP . . .

v=0\r\n
o=CiscoSystemsCCM-SIP 2000 3 IN IP4 10.10.10.1\r\n
s=SIP Call\r\n
c=IN IP4 10.10.10.200\r\n
t=0 0\r\n
[1] m=audio 16007 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
a=ptime:20\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n

```

```
[2] m=video 0 RTP/AVP 112 113 114\r\n
    b=TIAS:4000000\r\n
    a=rtpmap:112 H264/90000\r\n
    a=fmtp:112
profile-level-id=4D8028;packetization-mode=1;max-mbps=243000;max-fs=8100\r\n
    a=rtpmap:113 H264/90000\r\n
    a=fmtp:113 profile-level-id=42400D;packetization-mode=1;max-mbps=11880;max-fs=396\r\n
    a=rtpmap:114 H264/90000\r\n
    a=fmtp:114 profile-level-id=42400D;packetization-mode=0;max-mbps=11880;max-fs=396\r\n
    a=sendrecv\r\n
\r\n
```

insertMediaDescription(media-level, media-description)

```
insertMediaDescription(media-level, media-description)
```

この API は、指定したメディア レベルのメディア記述を追加するのに使用します。

例：メッセージメディア記述を、それが SDP 内の最初のメディア記述になるように挿入します。

スクリプト

```
M = {}
function M.outbound_INVITE(msg)
    local sdp = msg:getSdp()
    if sdp
    then
        local message = "m=message 1234 sip:alice@10.10.10.100\r\n"
        sdp = sdp:insertMediaDescription(1, message)
        msg:setSdp(sdp)
    end
end
return M
```

正規化前のメッセージ

```
INVITE SDP . . .

v=0\r\n
o=- 2000 3 IN IP4 10.10.10.100\r\n
s=SIP Call\r\n
c=IN IP4 10.10.10.100\r\n
t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
    a=rtpmap:9 G722/8000\r\n
    a=ptime:20\r\n
    a=rtpmap:18 G729/8000\r\n
    a=ptime:20\r\n
    a=fmtp:18 annexb=no\r\n
    a=rtpmap:101 telephone-event/8000\r\n
    a=fmtp:101 0-1\r\n
```

正規化後のメッセージ

```
INVITE SDP . . .

v=0\r\n
o=- 2000 3 IN IP4 10.10.10.100\r\n
s=SIP Call\r\n
c=IN IP4 10.10.10.100\r\n
t=0 0\r\n
[1] m=message 1234 sip:alice@10.10.10.100\r\n
```

```
[2] m=audio 18884 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
aptime:20\r\n
a=rtpmap:18 G729/8000\r\n
aptime:20\r\n
a=fmtp:18 annexb=no\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
```

removeMediaDescription(media-level)

```
removeMediaDescription(media-level)
```

この API は、指定したメディア レベルのメディア記述を削除するのに使用します。

例：

「[addMediaDescription\(media-description\)](#)」に示されている例を参照してください。この例は、`removeMediaDescription` も使用しています。

removeMediaDescription(media-contains)

```
removeMediaDescription(media-contains)
```

この API は、`m` 行に `media-contains` と完全に一致する部分を持つ特定のメディア記述を削除するのに使用します。



(注)

複数の `m` 行が一致する場合は、`media-description` と一致する最初のメディア記述だけが削除されます。

例：

このスクリプトは、着信 INVITE からビデオ メディア記述を削除して、保存します。対応する発信応答については、このスクリプトは、格納されているビデオ メディア記述を取得し、ポートを 0 に設定し、そのメディア記述を発信 SDP に追加します。

スクリプト

```
M = {}
function M.inbound_INVITE(msg)
  local sdp = msg:getSdp()
  if sdp
  then
    local video = sdp:getMediaDescription("video")
    if video
    then
      --remove the video media description
      sdp = sdp:removeMediaDescription("video")
      --store video media description
      local context = msg:getContext()
      context["video"] = video
      msg:setSdp(sdp)
    end
  end
end

function M.outbound_ANY_INVITE(msg)
  local sdp = msg:getSdp()
  --assume other side is expecting video before audio when there is
```

```

--a video m-line in the SDP
if sdp
then
  local context = msg:getContext()
  local video = context["video"]
  if video
  then
    --set port to zero in SDP answer
    video = video:gsub("video %d* RTP", "video 0 RTP")
    sdp = sdp:addMediaDescription(video)
    msg:setSdp(sdp)
  end
end
end
return M

```

正規化前のメッセージ

INVITE SDP . . .

```

v=0\r\n
o=- 2000 3 IN IP4 10.10.10.100\r\n
s=SIP Call\r\n
c=IN IP4 10.10.10.100\r\n
t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
a=ptime:20\r\n
a=rtpmap:18 G729/8000\r\n
a=ptime:20\r\n
a=fmtp:18 annexb=no\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
[2] m=video 32068 RTP/AVP 112 113 114\r\n
b=TIAS:4000000\r\n
a=rtpmap:112 H264/90000\r\n
a=fmtp:112
profile-level-id=4D8028;packetization-mode=1;max-mbps=243000;max-fs=8100\r\n
a=rtpmap:113 H264/90000\r\n
a=fmtp:113 profile-level-id=42400D;packetization-mode=1;max-mbps=11880;max-fs=396\r\n
a=rtpmap:114 H264/90000\r\n
a=fmtp:114 profile-level-id=42400D;packetization-mode=0;max-mbps=11880;max-fs=396\r\n
a=sendrecv\r\n
\r\n

```

200 Ok SDP . . .

```

v=0\r\n
o=CiscoSystemsCCM-SIP 2000 3 IN IP4 10.10.10.1\r\n
s=SIP Call\r\n
c=IN IP4 10.10.10.200\r\n
t=0 0\r\n
[1] m=audio 16007 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
a=ptime:20\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
\r\n

```

正規化後のメッセージ

```

INVITE SDP . . .

v=0\r\n
o=- 2000 3 IN IP4 10.10.10.100\r\n
s=SIP Call\r\n
c=IN IP4 10.10.10.100\r\n
t=0 0\r\n
[1] m=audio 18884 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
aptime:20\r\n
a=rtpmap:18 G729/8000\r\n
aptime:20\r\n
a=fmtp:18 annexb=no\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
\r\n

200 Ok SDP . . .

v=0\r\n
o=CiscoSystemsCCM-SIP 2000 3 IN IP4 10.10.10.1\r\n
s=SIP Call\r\n
c=IN IP4 10.10.10.200\r\n
t=0 0\r\n
[1] m=audio 16007 RTP/AVP 9 18 101\r\n
a=rtpmap:9 G722/8000\r\n
aptime:20\r\n
a=rtpmap:101 telephone-event/8000\r\n
a=fmtp:101 0-1\r\n
[2] m=video 0 RTP/AVP 112 113 114\r\n
b=TIAS:4000000\r\n
a=rtpmap:112 H264/90000\r\n
a=fmtp:112
profile-level-id=4D8028;packetization-mode=1;max-mbps=243000;max-fs=8100\r\n
a=rtpmap:113 H264/90000\r\n
a=fmtp:113 profile-level-id=42400D;packetization-mode=1;max-mbps=11880;max-fs=396\r\n
a=rtpmap:114 H264/90000\r\n
a=fmtp:114 profile-level-id=42400D;packetization-mode=0;max-mbps=11880;max-fs=396\r\n
a=sendrecv\r\n
\r\n

```



CHAPTER 5

SIP パススルー API

Cisco Unified CM は、SIP コール処理の点では Business to Business User Application (B2BUA; ビジネス ツー ビジネス ユーザ アプリケーション) です。パススルー オブジェクトが、情報のコール レッグからコール レッグへの受け渡しを可能にする一連の API を提供します。

次の SIP パススルー API が使用できます。

- 「[addHeader](#)」
- 「[addHeaderValueParameter](#)」
- 「[addHeaderUriParameter](#)」
- 「[addRequestUriParameter](#)」
- 「[addContentBody](#)」

addHeader

```
addHeader(header-name, header-value)
```

このメソッドは、ヘッダーの文字列名と値を受け取り、その情報をトリガーされた発信メッセージに含めるためにパススルー オブジェクトに追加します。

例

透過性がなければ、Cisco Unified CM は着信 **x-nt-corr-id** ヘッダーを知らないため、これを無視します。その結果、これは着信 INVITE から削除され、発信 INVITE には含まれなくなります。

このヘッダーのパススルーを有効にするには、このヘッダーをスクリプトの先頭にある **allowHeaders** テーブルに含める必要があります。さらに、メッセージ処理の中でパススルー オブジェクトに明示的に追加する必要があります。

スクリプト

```
M = {}
M.allowHeaders = {"x-nt-corr-id"}
function M.inbound_INVITE(msg)
    local ntcorr-id = msg.getHeader("x-nt-corr-id")
    if ntcorr-id
    then
        pt = msg.getPassThrough()
        pt.addHeader("x-nt-corr-id", ntcorr-id)
    end
end
return M
```

着信メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
x-nt-corr-id: 000002bf0f15140a0a@000075447daf-a561119
.
```

発信メッセージ

```
INVITE sip:1234@10.10.10.2 SIP/2.0
.
x-nt-corr-id: 000002bf0f15140a0a@000075447daf-a561119
.
```

addHeaderValueParameter

```
addHeaderValueParameter(header-name, parameter-name [,parameter-value])
```

このメソッドは、ヘッダーの名前、パラメータ名、およびパラメータ値を受け取り、その情報をトリガーされた発信メッセージに含めるためにパススルー オブジェクトに追加します。

ヘッダー名とパラメータ名は、必須引数です。parameter-value は、省略可能です。

**(注)**

デフォルトでは、スクリプトのロジックとは関係なく **Contact** ヘッダー値パラメータが渡されます。ただし、次の場合は例外で、コールログ固有と見なされ、Cisco Unified CM で適切に生成されます。

- audio
- video

例

透過性がなければ、Cisco Unified CM は、**From** ヘッダー内の着信 **x-tag** を知らないため、これを無視します。その結果、これは着信 INVITE から削除され、発信 INVITE には含まれなくなります。

このヘッダー パラメータのパススルーを有効にするには、メッセージ処理の中でこれを明示的にパススルー オブジェクトに追加する必要があります。

スクリプト

```
M = {}
function M.inbound_INVITE(msg)
    local xtag = msg.getHeaderValueParameter("From", "x-tag")
    if xtag
    then
        pt = msg.getPassThrough()
        pt:addHeaderValueParameter("From", "x-tag", xtag)
    end
end
return M
```

着信メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
From: <sip:1000@10.10.10.58>;tag=0988bf47-df77-4cb4;x-tag=42
.
```

発信メッセージ

```
INVITE sip:1234@10.10.10.2 SIP/2.0
.
From: <sip:1000@10.10.10.1>;tag=abcd;x-tag=42
.
```

addHeaderUriParameter

```
addHeaderUriParameter(header-name, parameter-name [,parameter-value])
```

このメソッドは、ヘッダーの名前、URI パラメータ名、およびパラメータ値を受け取り、その情報をトリガーされた発信メッセージに含めるためにパススルー オブジェクトに追加します。

ヘッダー名とパラメータ名は、必須引数です。パラメータの値は、省略可能です。

例：

透過性がなければ、Cisco Unified CM は、Contact ヘッダー URI 内の着信 **cca-id** パラメータを知らないため、これを無視します。その結果、これは着信 INVITE から削除され、発信 INVITE には含まれなくなります。

このヘッダー URI パラメータのパススルーを有効にするには、メッセージ処理の中でこれを明示的にパススルー オブジェクトに追加する必要があります。

**(注)**

この例では、パラメータは、発信メッセージ内で異なる名前を取ります（つまり、発信メッセージ内の発信 **cca-id** と、着信メッセージ内の **cca-id**）。

スクリプト

```
M = {}
function M.inbound_INVITE(msg)
  local occaid = msg.getHeaderUriParameter("Contact", "cca-id")
  if occaid
  then
    pt = msg.getPassThrough()
    pt.addHeaderUriParameter("Contact", "originating-cca-id", occaid)
  end
end
return M
```

着信メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
.
Contact: <sip:1000@10.10.10.58;cca-id=LSC.dsn.mil>
.
```

発信メッセージ

```
INVITE sip:1234@10.10.10.2 SIP/2.0
.
Contact: <sip:1000@10.10.10.1;originating-cca-id=LSC.dsn.mil>
.
```

addRequestUriParameter

```
addRequestUriParameter(parameter-name [,parameter-value])
```

このメソッドは、URI パラメータ名とパラメータ値を受け取り、その情報をトリガーされた発信メッセージに含めるためにパススルー オブジェクトに追加します。

パラメータ名は、必須引数です。パラメータの値は、省略可能です。



(注)

デフォルトでは、スクリプトのロジックとは関係なく初期 INVITE 内の要求 URI パラメータが渡されます。ただし、次の場合は例外で、コールログ固有と見なされ、CUCM で適切に生成されます。

- phone-context
- trunk-context
- tgrp
- user

例：着信ログが、パラメータを生成してから、発信要求 URI 内に格納するために受け渡します。

スクリプト

```
M = {}
function M.inbound_INVITE(msg)
    pt = msg:getPassThrough()
    pt:addRequestUriParameter("from-network", "service-provider")
end
return M
```

着信メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0
```

発信メッセージ

```
INVITE sip:1234@10.10.10.2;from-network=service-provider SIP/2.0
```

addContentBody

```
addContentBody(content-type, content-body [,content-disposition [,content-encoding
, [content-language]])])
```

このメソッドは、content-type、content-body、content-disposition、content-encoding、および content-language を受け取り、その情報をトリガーされた発信メッセージに含めるためにパススルー オブジェクトに追加します。

content-type と content-body は、必須引数です。content-disposition、content-encoding、および content-language は、省略可能なパラメータです。これらの値のいずれかが空または nil に指定された場合、ヘッダーがコンテンツの一部として含まれることはなくなります。

例：

透過性がなければ、Cisco Unified CM は着信 INFO メッセージとコンテンツ本文を無視します。透過性を使用した場合、Cisco Unified CM は Nortel PBX によって送信された独自コンテンツ本文を抽出し、コンテンツ本文から DTMF の数字を抽出して、新しい dtmf-relay コンテンツ本文を作成し、それを受け渡します。

スクリプト

```
M = {}
function M.inbound_INFO(msg)
    local body = msg:getContentBody("application/vnd.nortelnetworks.digits")
    if body
    then
        local digits = string.match(body, "d=(%d+)")
        if digits
        then
            pt = msg:getPassThrough()
            body = string.format("Signal=%d\r\nDuration=100\r\n", digits)
            pt:addContentBody("application/dtmf-relay", body)
        end
    end
end
return M
```

着信メッセージ

```
INFO sip: 1000@10.10.10.1 SIP/2.0
Via: SIP/2.0/UDP 10.10.10.57:5060
From: <sip:1234@10.10.10.57>;tag=d3f423d
To: <sip:1000@10.10.10.1>;tag=8942
Call-ID: 312352@10.10.10.57
CSeq: 5 INFO
Content-Type: application/vnd.nortelnetworks.digits
Content-Length: 72
```

```
p=Digit-Collection
y=Digits
s=success
u=12345678
i=87654321
d=4
```

発信メッセージ

```
INFO sip: 1000@10.10.10.58 SIP/2.0
Via: SIP/2.0/UDP 10.10.10.1:5060
From: <sip:1234@10.10.10.1>;tag=ef45ad
To: <sip:1000@10.10.10.58>;tag=1234567
Call-ID: 475623@10.10.10.1
CSeq: 5 INFO
Content-Type: application/dtmf-relay
Content-Length: 26
```

```
Signal=4
Duration=100
```




CHAPTER 6

SIP ユーティリティ API

Lua 環境には、データ文字列を操作するための API が用意されています。次の SIP ユーティリティ API が使用可能です。

- [「parseUri」](#)

parseUri

```
parseUri(uri)
```

この関数は、URI 文字列を受け取って、指定された URI を解析してから sipUri オブジェクトに格納し、その sipUri オブジェクトを呼び出し側のスクリプトに返します。有効な URI が指定されなかった場合は、Null が返されます。

例：

スクリプト

```
MM = {}  
function M.inbound_INVITE(msg)  
    local uriString = msg:getUri("P-Asserted-Identity")  
    if uriString  
    then  
        local uri = sipUtils.parseUri(uriString)  
    end  
end  
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
P-Asserted-Identity: <sip:1234@10.10.10.1>  
.
```

出力/結果

```
Local variable uriString is set to "sip:1234@10.10.10.1"  
Local variable uri is a sipUri object containing the parsed form of uriString  
"uri:getUser() is "1234"  
"uri:getHost() is "10.10.10.1"
```




CHAPTER 7

SIP URI API

Lua 環境には、解析済み SIP URI を操作するための一連の API が用意されています。次の SIP URI API が使用できます。

- [「applyNumberMask」](#)
- [「getHost」](#)
- [「getUser」](#)
- [「encode」](#)

applyNumberMask

```
applyNumberMask(mask, mask-char)
```

この関数は、番号マスクを受け取り、指定された番号マスクを解析済み URI のユーザ部に適用してから、修正されたユーザを解析済み sipUri オブジェクトに格納します。修正された URI のユーザ部を表す文字列が返されます。

番号マスクの適用

マスク パラメータは、URI のユーザ部に適用される変換を定義します。大文字の「X」は、番号マスクのワイルドカード部分を指定します。たとえば、マスクが「+1888XXXXXXX」の場合、例のユーザ部「4441234」にマスクが適用されたたすると、得られる文字列は「+18884441234」です。

マスクされるユーザ部に見つかった文字数が、マスクのワイルドカード文字数よりも少ない場合、最も左にあるワイルドカード文字は「X」で残されます。前述のマスクを例のユーザ部「1234」に適用すると、結果の文字列は「+1888XXX1234」になります。マスクされるユーザ部に見つかった文字数が、マスクのワイルドカード文字数よりも多い場合、ユーザ部の最も左にある文字が切り捨てられます。たとえば、マスク「+1888XXXX」がユーザ部「4441234」に適用される場合、結果の文字列は「+18881234」になります。

例：P-Asserted-Id エンティティの URI のユーザ部を、番号マスクを適用した後にローカル変数に値として格納します。

スクリプト

```
M = {}  
function M.inbound_INVITE(msg)  
    local uriString = msg:getUri("P-Asserted-Identity")  
    if uriString  
    then  
        local uri = sipUtils.parseUri(uriString)  
        if uri
```

```

        then
            local user = uri:applyNumberMask("+1919476XXXX")
        end
    end
end
return M

```

メッセージ

```

INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: <sip:1234@10.10.10.1>
.

```

出力/結果

```
Local variable user is set to "+19194761234"
```

getHost

```
getHost()
```

この関数は、解析した sipUri オブジェクトのホスト部分を取り出して、それを文字列として呼び出し側に返します。

例：P-Asserted-Identity ヘッダー内の URI のホスト部分をローカル変数に値として格納します。

スクリプト

```

M = {}
function M.inbound_INVITE(msg)
    local uriString = msg:getUri("P-Asserted-Identity")
    if uriString
    then
        local uri = sipUtils.parseUri(uriString)
        if uri
        then
            local host = uri:getHost()
        end
    end
end
return M

```

メッセージ

```

INVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: <sip:1234@10.10.10.1>
.

```

出力/結果

```
Local variable host is set to "10.10.10.1"
```

getUser

```
getUser()
```

この関数は、解析した sipUri オブジェクトのユーザ部分を取り出して、それを文字列として呼び出し側に返します。

例：P-Asserted-Identity ヘッダー内の URI のユーザ部分をローカル変数に値として格納します。

スクリプト

```
M = {}
function M.inbound_INVITE(msg)
    local uriString = msg:getUri("P-Asserted-Identity")
    if uriString
    then
        local uri = sipUtils.parseUri(uriString)
        if uri
        then
            local user = uri:getUser()
        end
    end
end
return M
```

メッセージ

```
IINVITE sip:1234@10.10.10.1 SIP/2.0
.
P-Asserted-Identity: <sip:1234@10.10.10.1>
.
```

出力/結果

```
Local variable user is set to "1234"
```

encode

```
encode()
```

この関数は、解析した sipUri オブジェクトを文字列にエンコードして、それを呼び出し側に返します。エンコードに先立って解析済み sipUri オブジェクトに加えられた変更はすべて、得られる文字列に反映されます。

例：P-Asserted-Identity ヘッダーから得た URI を解析し、番号マスクを適用してから、得られた URI をエンコードします。

スクリプト

```
M = {}
function M.inbound_INVITE(msg)
    local uriString = msg:getUri("P-Asserted-Identity")
    if uriString
    then
        local uri = sipUtils.parseUri(uriString)
        if uri
        then
            uri:applyNumberMask("+1919476XXXX")
            uriString = uri:encode()
        end
    end
end
```

```
    end  
end  
return M
```

メッセージ

```
INVITE sip:1234@10.10.10.1 SIP/2.0  
.  
P-Asserted-Identity: <sip:1234@10.10.10.1>
```

出力/結果

```
Local variable uriString is set to "<sip:+19194761234@10.10.10.1>"
```



CHAPTER 8

トレース API

スクリプトのトレーシングにより、スクリプト作成者がスクリプト内からトレースを生成することが可能になります。これは、スクリプトのデバッグだけに使用してください。スクリプトを稼働させ始めるときには、無効にする必要があります。スクリプトのトレーシングを有効にする方法は2通りあります。

1. 設定を使用する：SIP トランクに関連した [Script Trace] チェックボックスを参照してください。
2. スクリプトを使用する：`trace.enable()` API を参照してください。

次の表に、上述の2つの設定のすべての組み合わせにおいて、トレースがスクリプトによって生成されるかどうかを示します。

設定	スクリプト	トレースの生成
有効	有効	あり
有効	無効	なし
無効	有効	なし
無効	無効	なし

トレース ライブラリには、次の API があります。

- 「[trace.format](#)」
- 「[trace.enable](#)」
- 「[trace.disable](#)」
- 「[trace.enabled](#)」

trace.format

```
trace.format(format-string, p1, p2, ...)
```

この関数は、フォーマット文字列と一連のパラメータを受け取って、スクリプトに関連付けられているトランクのトレーシングが有効になっており、スクリプトが `trace.enable()` によってトレーシングを有効にしていた場合に、パラメータをフォーマット文字列に挿入し、その出力を SDI トレース ファイルに書き出します。

例

```
M = {}
trace.enable()
function M.inbound_INVITE(msg)
    local callid = msg.getHeader("Call-Id")
    trace.format("Call-Id header is %s", callid)
end
return M
```

trace.enable

```
trace.enable()
```

この関数は、スクリプトがトレーシングをローカルに有効にすることを可能にします。



(注)

トレーシングは、そのスクリプトを使用する SIP トランクの設定でも有効になっている必要があります。

例

前述の例、「[trace.format](#)」を参照してください。

trace.disable

```
trace.disable()
```

この関数は、スクリプトがトレーシングをローカルに無効にすることを可能にします。



(注)

関連付けられているトランクのスクリプト トレース設定フラグがオンになっている場合は、たとえここで無効にしても、**Trace.output** によってスクリプト トレースが生成されます。

例：

```
M = {}
trace.disable()
function M.inbound_INVITE(msg)
    local callid = msg.getHeader("Call-Id")
    trace.format("Call-Id header is %s", callid)
end
return M
```

trace.enabled

```
trace.enabled() turns a boolean
```

スクリプトのこのインスタンスに関連付けられているデバイス上でスクリプト トレーシングが有効になっているか、またはスクリプト自身でトレーシングを有効にしていた場合に、**true** が返されます。トレーシングが有効になっていない場合は、**false** が返されます。

例

```
M = {}
trace.enable()
function M.inbound_INVITE(msg)
    if trace.enabled()
    then
        local callid = msg.getHeader("Call-Id")
        trace.format("Call-Id header is %s", callid)
    end
end
return M
```



CHAPTER 9

スクリプト パラメータ API

スクリプト パラメータは、スクリプト作成者がトランク固有の設定パラメータ値を取得することを可能にします。

scriptParameters ライブラリには、次の API があります。

- [「getValue」](#)

getValue

```
getValue (parameter-name)
```

この関数は、パラメータ名を受け取って、パラメータの値を返します。指定された名前を持つパラメータが存在しており、値を持っていれば、同じ値が返されます。

名前は存在するものの、関連付けられている値がない場合は、空の文字列（引用符なし）が返されます。指定された名前を持つパラメータが存在しない場合は、nil が返されます。

例: 発信 INVITE にこのスクリプトを使用する SIP トランクに対して CCA-ID パラメータが設定されません。

スクリプト

```
M = {}
local ccaid = scriptParameters.getValue("CCA-ID")
function M.outbound_INVITE(msg)
    if ccaid
    then
        local contact = msg.getHeader("Contact")
        local replace = string.format("%s;CCA-ID=%s>", "%1", ccaid)
        contact = string.gsub(contact, "<sip:.*>", replace)
        msg.modifyHeader("Contact", contact)
    end
end
return M
```

正規化前のメッセージ

```
INVITE sip:1234@10.10.10.58 SIP/2.0
.
Contact: <sip:1000@10.10.10.1>
.
```

Message After Normalization

```
INVITE sip:1234@10.10.10.58 SIP/2.0
.
Contact: <sip:1000@10.10.10.1;CCA-ID=LSCSUB.dsn.mil>
```




CHAPTER 10

SIP 透過性

Cisco Unified CM は、Business to Business User Application (B2BUA; ビジネス ツー ビジネス ユーザ アプリケーション) です。このため、SIP から SIP への任意のコールが、2つの SIP ダイアログで構成されます。これは、しばしばダイアログが継続している間に片方のダイアログから他方のダイアログへ情報を渡すのに便利です。これには、コールセットアップメッセージング、ミッドコールメッセージング、およびコール終了メッセージングが含まれます。前述のパススルー オブジェクトを使用すれば、1つの SIP ダイアログ (1つのコール レッグを表す) から別の SIP ダイアログへと情報の透過的なパススルーをトリガーできます。

現在のところ、透過性は、2つの SIP トランク間の INVITE ダイアログに制限されています。SIP 回線が含まれる場合、透過性はサポートされません。SUBSCRIBE ダイアログ、PUBLISH、アウトオブダイアログ REFER、アウトオブダイアログ Unsolicited NOTIFY はサポートされず、MESSAGE もサポートされません。

サポートされる機能

次のメッセージで、透過性がサポートされます。

- 開始 INVITE メッセージとその関連応答メッセージ
 - INVITE 応答メッセージ
 - 180 応答メッセージ
 - 183 応答メッセージ
 - 200 応答メッセージ
 - 4XX、5XX、6XX 応答メッセージ
- reINVITE メッセージとその関連応答メッセージ
- UPDATE メッセージ (UPDATE メッセージへの応答メッセージには透過性はサポートされていません)
- INFO メッセージ (INFO メッセージへの応答メッセージには透過性はサポートされていません)
- BYE メッセージ (BYE メッセージへの応答メッセージには透過性はサポートされていません)

次のメッセージでは、透過性はサポートされません。

- ACK メッセージ
- PRACK メッセージとその関連応答メッセージ
- Rrplaces を持つ INVITE メッセージとその関連応答メッセージ
- REFER メッセージとその関連応答メッセージ

通常、Cisco Unified CM は、次の情報（パラメータ、ヘッダー、およびコンテンツ本文）をローカルで処理します。これは、特定のコール レッグと関連しています。したがって、理解された SIP 情報はそこで使用され、他のコール レッグ（いずれにしても SIP でない場合もある）へと伝えられることはありません。これにより、Cisco Unified CM が、SIP と H.323、SIP と MGCP などのさまざまなプロトコルの相互運用をサポートすることが可能になります。Cisco Unified CM で理解されない SIP 情報は、通常は無視されます。

以降では、Cisco Unified CM で理解されて使用される情報のことを**既知**の情報と言い、Cisco Unified CM には理解されず使用もされない情報のことを**未知**の情報と言います。

次の情報が、透過的に通過できます。

- パラメータ
- 未知のヘッダー
- 未知のコンテンツ本文

既知のヘッダー

次に、既知のヘッダーを示します。

- Accept
- Accept-Contact
- Accept-Resource-Priority
- Alert-Info
- Allow
- Allow-Events
- Also
- Authorization
- Bridge-Participant-ID
- Call-ID
- Call-Info
- CC-Diversion
- CC-Redirect
- Contact
- Content-Disposition
- Content-ID
- Content-Length
- Content-Type
- CSeq
- Date
- Diversion
- Event
- Expires
- From
- Geolocation

- Geolocation-Error
- Join
- Max-Forwards
- Min-Expires
- Min-SE
- MIME-Version
- P-Asserted-Identity
- P-Preferred-Identity
- Privacy
- Proxy-Authenticate
- Proxy-Authorization
- Proxy-Require
- RAck
- Reason
- Recv-Info
- Refer-To
- Referred-By
- Reject-Contact
- Remote-Party-ID
- Replaces
- Request-Disposition
- Requested-By
- Require
- Resource-Priority
- Retry-After
- RSeq
- RTP-RxStat
- RTP-TxStat
- Server
- Session
- Session-Expires
- SIP-ETag
- SIP-If-Match
- Subject
- Subscription-State
- Supported
- Target-Dialog
- To

- Unsupported
- User-Agent
- Via
- Warning
- WWW-Authenticate
- X-Cisco-EMCCInfo
- X-Cisco-FallbackID
- X-Cisco-ViPR-Ticket

既知のコンテンツ本文

- application/sdp

スクリプトが既知のヘッダーまたはコンテンツ本文を受け渡そうとすると、実行エラーがトリガーされます。

スクリプト作成者は、既知のデータが Cisco Unified CM 処理にかけられたり、Cisco Unified CM 処理と干渉したりすることなく受け渡される方法があることをすぐに理解できるでしょう。実際、スクリプトは、既知のヘッダーから値を取得し、未知のヘッダーに格納することができます。コンテンツ本文についても、同じことが行えます。下に示す 181 透過性の例では、Reason ヘッダーでこれを行っています。この例は、Reason ヘッダーの値を取得し、それを X-Reason ヘッダーとして受け渡します。当然のことながら、相手サイドに X-Reason ヘッダーを消費して削除するスクリプトがなければ、このヘッダーはネットワークに送信されます。

181 透過性の例

透過性がなければ、Cisco Unified CM が発信トランク レッグで 181 を受信した場合、Cisco Unified CM のネイティブ動作は、着信トランク レッグ上で 180 を送り返すことです。181 透過性を達成するには、着信 181 (B サイドで受信) と予定発信 180 (A サイドで送信) の両方にスクリプトが必要です。

181 は最初に PBX-B から受信されるので、まず次のことを行うことを検討します。

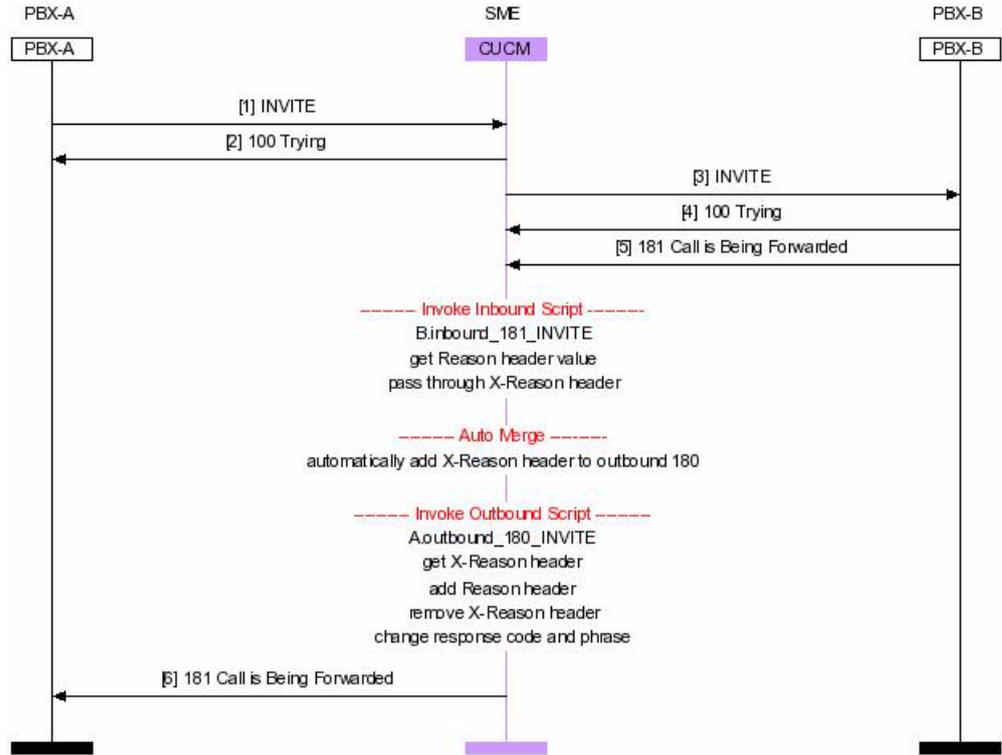
- Reason ヘッダーの値の取得
- Reason ヘッダーの値のパススルー : Reason ヘッダーは既知のヘッダーなので、スクリプトは、ヘッダー名 X-Reason を使用して値を受け渡すことにより、既知のヘッダーのチェックをバイパスします。

このパススルー データは、Cisco Unified CM によって自動的に、これから送信される発信メッセージとマージされます。前述のとおり、Cisco Unified CM は 180 をネイティブに送信します。このため、自動マージ機能により、X-Reason ヘッダーが発信 180 内に置かれます。

次に、次の内容を A サイドで行う必要があるかどうかを検討する必要があります。

- X-Reason ヘッダーの値の取得と、181 に関して何かを含んでいるかどうかの確認
- X-Reason ヘッダーの値を持つ Reason ヘッダーの追加
- X-Reason ヘッダーの削除
- 応答コードとフレーズの 181 Call is Being Forwarded への変換

これらの手順を、次のコールフロー図に示します。



B サイドと A サイドのスク립トを次に示します。

B サイドのスク립ト

```

B = {}
function B.inbound_181_INVITE(msg)
    local pt = msg:getPassThrough()
    local reason = msg:getHeader("Reason")
    if pt and reason
    then
        pt:addHeader("X-Reason", reason)
    end
end
return B
    
```

A サイドのスク립ト

```

A = {}
function A.outbound_180_INVITE(msg)
    local reason = msg:getHeader("X-Reason")
    if reason
    then
        if string.find(reason, "cause=181")
        then
            msg:setResponseCode(181, "Call is being forwarded")
            msg:addHeader("Reason", reason)
        end

        msg:removeHeader("X-Reason")
    end
end
    
```

```
end
return A
```

INFO 透過性の例

透過性がなければ、Cisco Unified CM は着信 INFO メッセージとコンテンツ本文を無視します。透過性を使用した場合、Cisco Unified CM は Nortel PBX によって送信された独自コンテンツ本文を抽出し、コンテンツ本文から DTMF の数字を抽出して、新しい dtmf-relay コンテンツ本文を作成し、それを他方のコール レッグへと受け渡します。

スクリプト

```
M = {}
function M.inbound_INFO(msg)
    local body = msg:getContentBody("application/vnd.nortelnetworks.digits")
    if body
    then
        local digits = string.match(body, "d=(%d+)")
        if digits
        then
            pt = msg:getPassThrough()
            body = string.format("Signal=%d\r\nDuration=100\r\n", digits)
            pt:addContentBody("application/dtmf-relay", body)
        end
    end
end
return M
```

着信メッセージ

```
INFO sip: 1000@10.10.10.1 SIP/2.0
Via: SIP/2.0/UDP 10.10.10.57:5060
From: <sip:1234@10.10.10.57>;tag=d3f423d
To: <sip:1000@10.10.10.1>;tag=8942
Call-ID: 312352@10.10.10.57
CSeq: 5 INFO
Content-Type: application/vnd.nortelnetworks.digits
Content-Length: 72
```

```
p=Digit-Collection
y=Digits
s=success
u=12345678
i=87654321
d=4
```

発信メッセージ

```
INFO sip: 1000@10.10.10.58 SIP/2.0
Via: SIP/2.0/UDP 10.10.10.1:5060
From: <sip:1234@10.10.10.1>;tag=ef45ad
To: <sip:1000@10.10.10.58>;tag=1234567
Call-ID: 475623@10.10.10.1
CSeq: 5 INFO
Content-Type: application/dtmf-relay
Content-Length: 26
```

```
Signal=4
Duration=100
```