開発者用 SIP 透過性および正規化ガイド

Cisco Unified Communications Manager Release 9.1(1)

Text Part Number: OL-28326-01-J

【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。

開発者用 SIP 透過性および正規化ガイド Cisco Unified Communications Manager 9.1(1)

CONTENTS

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

このマニュアルは、SIP の透過性および正規化を使用し、Cisco Unified Communications Manager 9.1(1) と統合するアプリケーションまたは製品を開発している、開発業者、ベンダー、およびお客様を対象に説明します。

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」	特定の配置環境での、セッション管理（SM）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 環境の概要について説明します。
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セッション記述プロトコル（SDP）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 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 メッセージ API について説明します。
第 4 章「SDP API」	セッション記述プロトコル（SDP）のコンテンツ本文に関連付けられた 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」	データ文字列の操作を可能にする 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「Trace 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または回線固有の設定パラメータ値を取得できるかについて説明します。
第 10 章「SIP の透過性」	SIP の透過性に使用される 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
太字	コマンドおよびキーワードは**太字**で示しています。
イタリック体	ユーザが値を指定する引数は、*イタリック体*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。
`太字の 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。
`イタリック体の screen フォント`	ユーザが値を指定する引数は、*`イタリック体の screen フォント`*で示しています。
→	このポインタは、例の中の重要な行を強調しています。
^	^ 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 「問題解決に役立つ情報」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、シスコの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（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。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

この章では、特定の配置環境での、セッション管理（SM）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、非独占の、動作が軽いスクリプト言語です。このマニュアルの対象者は、Lua スクリプト言語の基本的な知識があることが前提となります。

Cisco Unified CM では、SIP メッセージをカスタマイズするための SIP 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ます。

- 正規化：これは、着信および発信メッセージの変換プロセスです。
  - 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 により適合させ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、この目的に 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。
  - 発信メッセージの場合、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、メッセージを外部 SIP デバイス（例：別の SIP 対応 PBX）により適合させ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 トランクまたは SIP 回線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
- コンテンツ本文
- 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
- コンテンツ本文

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。

スクリプト管理の詳細については、『*Cisco Unified Communications Manager Administration Guide*』を参照してください。

スクリプト環境

Cisco Unified CM SIP トランクの動作をカスタマイズするためのインターフェイスは、Lua という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）に使用できる Lua 環境は Lua の限定的なサブセットです。Lua で提供される基本機能の他に、次のような機能があります。

- 単語表記
- 値とタイプ
- 変数
- ステートメント
- 式など

Lua では、次のような一部の基本ライブラリも提供されています。

- ベース
- 共通ルーチン
- モジュール
- 文字列操作
- テーブル操作
- 数学関数
- OS 機能
- IO 機能

- 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、次がサポートされます。

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

Cisco SIP Lua 環境では、使用するスクリプト用にグローバル環境があります。この環境では、スクリプトに対してデフォルトの Lua グローバル環境（つまり _G）は提供されません。

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 メッセージを操作します。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、シスコの SIP メッセージ ライブラリに定義されている API を使用して、**msg** パラメータにアクセスして操作を行います。

このマニュアルの以降では、**メッセージ ハンドラ**の構造の詳細について説明します。次の項にシスコの SIP メッセージ ライブラリ API についての詳細を説明します。

ここでは、発信 INVITE で「Cisco-Guid」ヘッダーを削除するだけの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。

簡単なスクリプト：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
```

上記のスクリプトに重要な部分が 3 つあります。

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：2 ～ 4 行目で、着信 INVITE の**メッセージ ハンドラ**が定義されます。このスクリプトに関連付けられた SIP トランクまたは SIP 回線で着信 INVITE が受信されたときに、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、History-Info ヘッダーが Diversion ヘッダーに変換されます。

   5 ～ 6 行目で、発信 INVITE の**メッセージ ハンドラ**が定義されます。このスクリプトに関連付けられた SIP トランクまたは SIP 回線で発信 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。

スクリプトで複数のメッセージ ハンドラを定義できます。メッセージ ハンドラの名前は、特定の SIP メッセージに対して呼び出されるメッセージ ハンドラがあれば、そのハンドラを示します。

3. モジュールを返す：最後の行はモジュールの名前を返します。この行は必須です。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 ハンドラを見つける方法です。

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 メッセージを操作します。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。

名前付け

メッセージ ハンドラの命名規則によって、特定の SIP メッセージに対して起動されるメッセージ ハンドラが示されます。仕様による各種 SIP メッセージは、*要求*と*応答*にわかれます。

- *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」にある名前です。

  ```
  <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>
  ```

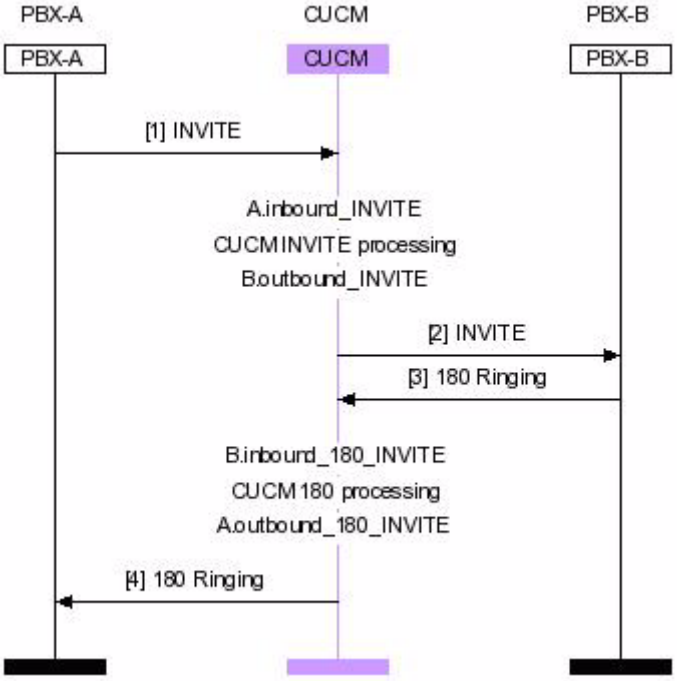
例

```
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 ハンドラ名は、ワイルドカードもサポートします。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 メッセージかによって異なります。

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 応答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、特定のメッセージの**メッセージ ハンドラ**を見つけます。

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 ハンドラが見つけます。

   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、これは最も明らかな一致のため、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、現在サポートされていません。

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
```

ただし、次のメッセージ ハンドラは無効です。

```
inbound_401_ANY
outbound_200_ANY
```

など

CHAPTER 2

SIP と SDP の正規化

Lua スクリプト作成環境では、SIP メッセージおよびそれに関連するすべてのセッション記述プロトコル（SDP）を操作するための基礎となるオブジェクトを確立します。スクリプトでは、これらのオブジェクトを考慮する必要はありません。次の API セットを使用してオブジェクトにアクセスします。

- SIP メッセージ API：これらの API を使用すると、スクリプトで、多様な方法によって 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、多様な方法によって 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、1 つのコール レッグから別のコール レッグへ情報を受け渡しできます。
- SIP ユーティリティ API：これらの API には、Uniform Resource Identifier（URI）の解析を SIP URI オブジェクトに組み込んだデータを処理するスクリプトのための役立つユーティリティが用意されています。
- SIP URI API：これらの API を使用すると、スクリプトで、解析済み SIP URI オブジェクトを操作できます。
- Trace API：これらの API を使用すると、スクリプトで、トレースの有効/無効の切り替え、トレースが有効かどうかの特定、およびトレースの生成を行うことができます。
- 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、トランクまたは回線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 API

Lua スクリプト作成環境には、メッセージの処理を可能にする一連の API が用意されています。これらの API について、次のカテゴリで説明します。

要求行または応答行での操作

getRequestLine

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

このメソッドは次の 3 つの値を返します。

- 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の値を設定します。

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場合、指定された URI パラメータの値を返されます。

- 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に対して起動されると、nil が返されます。

例：ローカル変数を、要求 URI のユーザ 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）の 3 つの値が返されます。この API が要求に対して起動されると、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。どちらの値にも nil を使用できます。値に nil が指定されている場合、メッセージに格納されている既存の値が使用されます。この API が要求に対して起動されると、実行時エラーがトリガーされます。

例：発信 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

```
allowHeaders is a Lua table specified by the script writer
```

Cisco Unified CM が認識していないヘッダーに対するデフォルト動作では、このようなヘッダーを無視します。場合によっては、このようなヘッダーを正規化時に使用したり、ヘッダーを透過的に受け渡したりすることが望ましいこともあります。**allowHeaders** テーブルでヘッダー名を指定することにより、スクリプトを使用して、Cisco Unified CM でのヘッダーの認識が可能になり、スクリプトで正規化のためにローカルで使用したり、透過的に受け渡したりすることは問題なく行えます。

例：

allowHeaders は `History-Info` を指定します。指定しない場合は、スクリプトの処理前に、着信 History-Info ヘッダーがドロップされます。したがって、**convertHIToDiversion** で有用な結果は得られません。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場合、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場合、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 のインデックス化は 1 から始まります。n 個のヘッダー値がある場合、最初の値はインデックス 1 であり、最後の値はインデックス n です。Lua の **ipairs** 関数では、テーブル全体で繰り返し使用できます

例：History-Info ヘッダーの値を含むローカル 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、ヘッダーの名前およびヘッダー パラメータが指定された場合、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場合、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、ヘッダーの文字列名および値が指定された場合、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場合、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対して複数のヘッダー値がある場合、最初のヘッダー値のみが変更されます。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、アクションは実行されません。

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

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場合、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対して複数のヘッダー値がある場合、最初のヘッダー値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、ヘッダーの文字列名および値が指定された場合、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実際似ています。

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場合、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 に適用されます。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。

次のいずれかの条件を満たす場合、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、一番左のワイルド カード文字から「X」が置かれます。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、ユーザ部の一番左の文字から切り捨てられます。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 ヘッダーの番号に番号マスクを適用します。

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。このような例では、リダイレクト番号の送信のために 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
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のために 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ことがあります。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。

（注） Diversion ヘッダーは、History-Info ヘッダーに「cause=」タグが存在する場合にだけ追加されます。以下の例では、Diversion ヘッダーが 1 つしかないが、History-Info ヘッダーが 2 つある場合、そのうちの 1 つには「cause=」タグがあることに注意してください。

例

allowHeaders では、「History-Info」を指定する必要があります。指定しない場合、スクリプトの処理前に、着信 History-Info ヘッダーがドロップされます。したがって、convertHIToDiversion へのコールで有用な結果は得られません。

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場合、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場合、メッセージからヘッダー値が削除されます。値がヘッダー値だけだった場合、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
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 が含まれるローカル変数を確立します（SDP がない場合は nil）。

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（つまり「\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 コンテンツ本文として保存します。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

前述のスクリプト実行後の最後の 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。メッセージに他のコンテンツ本文がない場合、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
```

コンテンツ本文の操作

- 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場合、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 メッセージのコンテンツ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 が指定された場合、メッセージにコンテンツ本文が追加され、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、および言語も指定できます。

例：発信 UPDATE メッセージに、プレーン テキストのコンテンツ本文を追加します。

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 が指定された場合、関連付けられているコンテンツ本文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 text/plain コンテンツ本文を削除します。

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()
```

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ものとして CUCM が動作します。発信メッセージのブロックでは、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できる 18X の場合、CUCM では block() への呼び出しは事実上無視します。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-c1fe2b0e03b6-21456114
.
.
```

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、ダイアログの継続中に使用される、多様なメッセージ ハンドラでのコンテキストからデータを保存および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慮す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、コンテキストを使用して、INVITE で受信した History-Info ヘッダーが保存されます。その INVITE への応答が送信されると、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」パラメータを使用して、後続の reINVITE 応答にこれらのヘッダーが追加されることを防止します。

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

```
isInitialInviteRequest() returns true or false
```

このメソッドでは、メッセージが要求であり、メソッドが INVITE であり、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 です。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 ではありません。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、メッセージが要求であり、メソッドが INVITE であり、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 です。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 ではありません。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

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

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場合、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：ローカル変数を、P-Asserted-Identity ヘッダーの 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

このセクションでは、セッション記述プロトコル（SDP）のコンテンツ本文に関連付けられた API について説明します。SIP メッセージ オブジェクトとは異なり、SDP は基本的な文字列として扱われます。これを使用して、スクリプト作成者は Lua 文字列ライブラリの機能を利用することができます。Lua 文字列ライブラリ関数に加えて、シスコは、SDP の操作を容易にするためにいくつかの API も提供します。Cisco API は文字列ライブラリに直接追加されます。

ただし、スクリプトで SDP を操作するには、まず SIP メッセージ オブジェクトで提供される **getSdp()** API を使用して、Lua SIP メッセージ 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、スクリプトでシスコの API を含む文字列ライブラリを使用して、SDP を操作できます。変更すると、SIP メッセージ オブジェクトによって提供される **setSdp(sdp)** API を使用して、SIP メッセージ オブジェクトに SDP が書き戻されます。これらの API に関する詳細については、SIP メッセージ 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: etSdp(sdp)
```

SDP を操作するために、シスコでは、個々の回線の取得、変更、追加、挿入、削除のための API、ならびにすべてのメディア記述を提供します。この章の以降の部分では、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 を作用するように対処することが求められます。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。不可能ではないにしても、無効な SDP を作用させるのは簡単ではありません。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

次の行レベルの 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 では、sdp パラメータの SDP 構成を強制することはなく、文字列同様に扱います。スクリプト作成者は、API を使用して、SDP の一部分（たとえば、音声メディア記述など）を操作できます。
start-of-line	**start-of-line** パラメータは、sdp 内の行の先頭と一致させるために使用する文字列です。一般的なスクリプトでは、たとえば、「o=」「a=rtpmap:9」を使用して、この文字列で始まる SDP の最初の行を取得 / 変更 / 挿入 / 削除できます。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」パラメータで始まる行内で検索する、追加の一致条件として使用される文字列です。nil が指定されている場合、**line-contains** は一致条件としては使用されません。その場合、**start-of-line** パラメータだけが検索に使用されます。 **line-contains** が nil でない場合、**start-of-line** と **line-contains** の両方が検索に使用されます。（両方）検索条件に一致する SDP の行が複数ある場合、最初の一致だけが返されます。 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

上記の API では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
```

このメソッドでは、sdp 内の **start-of-line** で開始する行のうち、文字列 **line-contains** を含む最初の行が返されます。

例：ローカル変数を、SDP からの c 行の値に設定します。

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、**start-of-line** で始まる「sdp」の最初の行を検出します。**line-contains** パラメータが nil でない場合は、その値も行内に存在している必要があります。一致する行（行終了文字を含む）は **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呼び出し側で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なしの 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
a=ptime: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
```

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
a=ptime: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呼び出し側で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)
```

このメソッドでは、**line-contains** パラメータが指定されている場合、sdp 内の **start-of-line** で開始する行のうち、**line-contains** も含む最初の行を検出します。次に、見つかった行の後に **new-line** を挿入します。

例：SDP 内の「a=」および「G729」を持つ行の後に 1 行挿入します。

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)
```

このメソッドでは、**line-contains** パラメータが指定されている場合、sdp 内の **start-of-line** で開始する行のうち、**line-contains** も含む最初の行を検出します。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 1 行挿入します。

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
a=ptime:20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
```

removeLine

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

このメソッドでは、「line-contains」パラメータが指定されている場合、「sdp」内の「start-of-line」で開始する行のうち、「line-contains」を含む最初の行を検出します。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
a=ptime: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
```

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 から始まります。したがって、最初のメディア記述はインデックス 1 です。2 つ目はインデックス 2 となり、以降も同様です。

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（「**\r**」、「**\n**」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 です。2 つ目はメディア レベル 2 で、以降も同様です。
media-contains	media-contains パラメータは、m 行内で一致対象を（例：メディア タイプ）見つけるために使用する文字列です。 たとえば、audio、video、message などを使用して、完全に一致する media-contains 値を m 行に含む、最初に一致するメディア記述を取得、修正、または削除できます。
media-description	media-description パラメータは、完全なメディア記述を示す文字列です。複数の行からなり必要な埋め込み制御文字（「\r」および「\n」）を含みます。 最後の行は「\r\n」で終了させることを強く推奨します。ただし、これらの API が行う内容は純粋なテキスト操作で、テキストの形式に関して強制されるルールは一切ありません。実際に、実装によっては、各行の末尾に「\r」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、いずれのケースにも対処するために必要な柔軟性を提供し、さらに 2 つのスタイル間で簡単に変換することもできます（たとえば、「\r\n」を「\n」に変換したり、その逆に変換したりすることはそれほど問題ではありません）。 次の文字列は、完全なメディア記述の一例です。 `"m=audio 18884 RTP/AVP 9\r\na=rtpmap:9 G722/8000\r\n"`

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 から始まります

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：ローカル変数を、最初のメディア記述の値に設定します。

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返されます。

例：ローカル変数を、m 行に「audio」というテキストを含む最初のメディア記述の値に設定します。

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
     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 audio 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
```

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」を含む行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のに使用します。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ることに注意してください。

例：「a=ptime:20\r\n」を含む行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 に設定して、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
     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
```

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) で提供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 に設定して、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
     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
```

CHAPTER 5

SIP パススルー API

Cisco Unified CM は、SIP コール処理の点ではビジネス ツー ビジネス ユーザ アプリケーション（B2BUA）です。パススルー オブジェクトでは、コール レッグからコール レッグに情報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および値が指定された場合、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

透過性がない場合、Cisco Unified CM は着信 **x-nt-corr-id** ヘッダーを認識していないため、これを無視します。実際に、これは着信 INVITE から削除され、発信 INVITE には含まれません。

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 ntcorrid = msg:getHeader("x-nt-corr-id")
    if ntcorrid
    then
        pt = msg:getPassThrough()
        pt:addHeader("x-nt-corr-id", ntcorrid)
    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が指定された場合、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、**Contact** ヘッダー値のパラメータは、スクリプトのロジックとは無関係に渡されます。ただし、次の場合は例外で、コール レッグ固有であり、Cisco Unified CM で適切に生成されたものと見なされます。

- audio
- video

例

透過性がない場合、Cisco Unified CM は、**From** ヘッダー内の着信 **x-tag** を認識していないため、これを無視します。実際に、これは着信 INVITE から削除され、発信 INVITE には含まれません。

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が指定された場合、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。パラメータ値は、省略可能です。

例：

透過性がない場合、Cisco Unified CM は、Contact ヘッダー URI 内の着信 **cca-id** パラメータを認識していないため、これを無視します。実際に、これは着信 INVITE から削除され、発信 INVITE には含まれません。

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、パラメータは、発信メッセージでは別の名前を持ちます（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が指定された場合、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 レッグ固有であり、CUCM で適切に生成されたものと見なされます。

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

例：着信レッグでパラメータを生成し、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 が指定された場合、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 および 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なくなります。

例：

透過性がない場合、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。透過性を使用して、Cisco Unified CM は Nortel PBX によって送信された独自のコンテンツ本文を抽出します。さらにコンテンツ本文から DTMF ディジットを抽出して、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、データ文字列の操作を可能にする 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 文字列が指定された場合、指定された URI を解析して sipUri オブジェクトに組み込み、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 の操作を可能にする一連の 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、番号マスクが指定された場合、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 のユーザ部の文字列表現が返されます。

番号マスクの適用

マスク パラメータによって、URI のユーザ部に適用される変換が定義されます。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とすると、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、一番左のワイルドカード文字から「X」が置かれます。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、ユーザ部の一番左の文字から切り捨てられます。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：番号マスクを URI のユーザ部に適用した後、ローカル変数を、P-Asserted-Identity の URI のユーザ部の値に設定します

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、解析済み 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：ローカル変数を、P-Asserted-Identity ヘッダー内の URI のホスト部分に設定します。

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、解析済み 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：ローカル変数を、P-Asserted-Identity ヘッダー内の URI のユーザ部分に設定します

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、解析済み 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。エンコードよりも前に解析済み 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 ヘッダーの 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
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

Trace API

スクリプト トレーシングを使用して、スクリプト作成者がスクリプト内からトレースを生成できます。これは、スクリプトのデバッグだけに使用してください。スクリプトの動作を開始する場合は、これを無効にする必要があります。スクリプト トレーシングを有効にする方法は 2 通りあります。

1. 設定による方法：SIP トランクまたは SIP プロファイルに関連付けられた [スクリプト トレース（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	生成されるトレース
有効	有効	Yes
有効	無効	No
無効	有効	No
無効	無効	No

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パラメータのリストが指定された場合、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いて、そのスクリプトで **trace.enable()** によってトレーシングが有効にされていれば、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、スクリプトでトレーシングをローカルに有効にできます。

（注）　トレーシングは、スクリプトを使用して、SIP トランクまたは SIP プロファイルの設定によって有効になっている必要もあります。

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、スクリプトでトレーシングをローカルに無効にできます。

（注）　関連付けられている SIP トランクまたは SIP プロファイルのスクリプト 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 パラメータによって、スクリプト作成者がトランク固有または回線固有の設定パラメータ値を取得できます。

scriptParameters ライブラリには次の API があります。

- getValue

getValue

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

この関数では、パラメータ名が指定された場合、パラメータの値が返されます。指定された名前のパラメータが存在し、値を持つ場合は、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。指定された名前でこのようなパラメータがない場合、nil が返されます。

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

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>
.
```

正規化後のメッセージ

```
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 の透過性

Cisco Unified Communications Manager（Unified CM）は Back to Back User Agent（B2BUA）です。このため、SIP 間の任意のコールは 2 つの SIP ダイアログで構成されます。これは、多くの場合、ダイアログの継続中にダイアログ間で情報を渡すときに役立ちます。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あります。前述のパススルー オブジェクトを使用して、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、SIP トランクまたは SIP 回線の INVITE ダイアログに制限されています。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。

- 初期 INVITE および関連応答
  - INVITE 応答
  - 180 応答
  - 183 応答
  - 200 応答
  - 4XX、5XX、6XX 応答
- reINVITE および関連応答
- UPDATE メッセージ（UPDATE への応答に対して透過性はサポートされていません）
- INFO メッセージ（INFO への応答に対して透過性はサポートされていません）
- BYE メッセージ（BYE への応答に対する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
- PRACK および関連応答
- Replaces を持つ INVITE および関連応答
- REFER と関連応答

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）に渡されることはあり

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 インターワーキングをサポートできます。Cisco Unified CM によって理解されていない SIP 情報は通常無視されます。

次の項では、Cisco Unified CM によって理解され使用された情報のことを、**既知**であると、Cisco Unified CM によって理解されず、使用されない情報のことを、**不明**であると表現します。

次の情報は透過的に通過できます。

- パラメータ
- 不明なヘッダー
- 不明なコンテンツ本文

既知のヘッダー

次に、既知のヘッダーのリストを示します。

- 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

スクリプトで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。

スクリプト作成者であれば、既知のデータが Cisco Unified CM の処理にかけられたり、Cisco Unified CM の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することができます。実際、スクリプトは、既知のヘッダーの値を取得し、不明な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、相手側に X-Reason ヘッダーを使用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。

いずれの場合も、既知のヘッダーまたはコンテンツ本文の透過性はサポートされておらず、SIP の透過性と正規化はこの目的のためのものではありません。SIP の透過性を使用して、既知のヘッダーまたはコンテンツ本文を受け渡す場合、その結果は保証されません。

特に、SDP の透過性はサポートされていません。Cisco Unified CM は、この場合、リージョン帯域幅ポリシーまたはコール アドミッション制御を適用できません。また必要となる可能性のあるメディア リソースを挿入することもできません。多数のコール フローにより、SDP の更新が発生するため、Cisco Unified CM を正しく実行できません。SDP で宣言した要素を正常に操作できる可能性はありますが、ネゴシエートされた要素の操作は初期コール セットアップよりも成功する可能性が低くなります。

181 透過性の例

透過性がない場合、Cisco Unified CM が発信トランク レッグで 181 を受信すると、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。

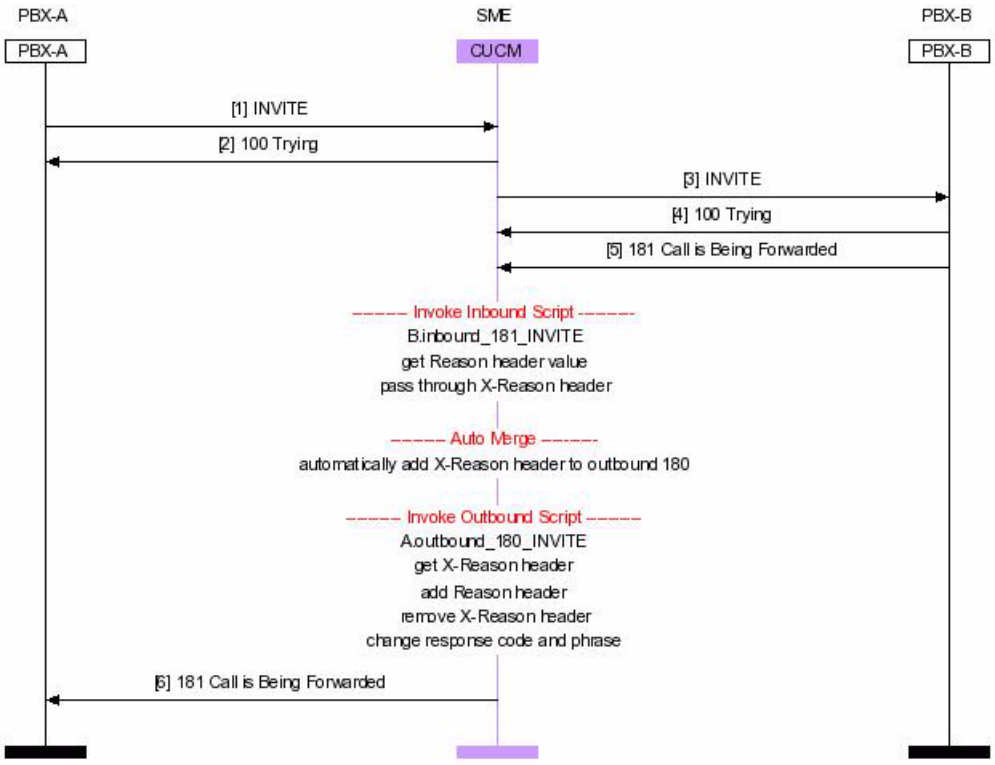
このパススルー データは、Cisco Unified CM によって自動的に、これから送信される発信メッセージとマージされます。前述のとおり、Cisco Unified CM は 180 をネイティブに送信します。このため、自動マージ機能により、X-Reason ヘッダーが発信 180 内に置かれます。

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 サイドのスクリプトを次に示します。

B サイドのスクリプト

```
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 サイドのスクリプト

```
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
return A
```

INFO 透過性の例

透過性がない場合、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。透過性を使用して、Cisco Unified CM は Nortel PBX によって送信された独自のコンテンツ本文を抽出します。そのコンテンツ本文から DTMF ディジットを抽出し、新しい dtmf-relay コンテンツ本文を作成して、それを相手側のコール 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
```

PCV および PAI ヘッダーのスクリプト パラメータ

P-Charging Vector の場合

P-Charging Vector ヘッダーのスクリプト パラメータは、モバイル関連情報を受け渡すための透過性を追加するように拡張されています。

リリース 8.6(1) よりも前は、Cisco Unified Communications Manager が P-Charging Vector を持つコールを受信した場合、Cisco Unified Communications Manager は、モバイルまたは IP Multimedia Subsystem（IMS）値なしの着信トランク レッグのコール情報を、そのサービス プロバイダーに送信していました。現在は、変更された PCV スクリプト パラメータを使用してより適切な透過性を得るため、Cisco Unified Communications Manager はモバイルまたは PSTN から送信された課金情報を抽出し（B サイドで受信）、その情報を変更せずに相手側のコール レッグに受け渡しします（A サイドで送信）。

スクリプト パラメータ

term-ioi：このパラメータでは、Call Manager から発信される発信 200 OK の P-Charging-Vector ヘッダーに term-ioi パラメータを、スクリプトで追加する必要があるかどうかを設定します。

P-Asserted Identity の場合

P-Asserted Identity ヘッダーのスクリプト パラメータは、Cisco Unified Communications Manager が、Cisco Unified Communications Manager から発信された発信コールに対する着信コールに存在する、未修正の PAI 情報を受け渡しできるように拡張されました。

スクリプト パラメータ

pai-passthru：このパラメータでは、発信コールの P-Asserted-Identity を、スクリプトで受け渡す必要があるかを設定します。

転送カウンタ

転送カウンタ スクリプトは、コール転送シナリオの転送カウンタ パラメータを処理します。転送カウンタ スクリプトは次の機能を提供します。

- Unified CM クラスタ内の転送なしの基本的なタンデム（トランク間）コールのパススルー転送カウンタ パラメータとして動作します。
- Unified CM クラスタ内で複数の転送を含むタンデム コールの転送カウンタ パラメータを処理します。

- 着信 INVITE メッセージに Diversion ヘッダーが 3 つ以上あり、さらに Unified CM クラスタ内に複数の転送がある場合、転送カウンタ パラメータを処理します。

例外

転送カウンタ スクリプトには次の例外がります。

- 転送カウンタ スクリプトはタンデム コールだけに適用されます。
- クラスタ内に複数の転送がある場合、転送カウンタ パラメータは 1 ずつ加算されます。

スクリプト

```
--[[
    Description:
        Diversion Counter Handling For Call Forward Scenarios. This script should be
attached to both inbound and outbound trunk. This script provides the following
functionalists:

       1. Pass through counter parameter in case Basic Tandem call with NO Diversion
within the cluster.
       2. Handling of counter parameter for Tandem call with one or more Diversion within
CUCM cluster
       3. Handling of counter parameter for Tandem call when inbound INVITE has more than
two Diversion headers and there is also one or more Diversion within CUCM cluster

    Exceptions:
       1. This script is only applicable for Tandem (trunk-to-trunk) calls.
       2. If there are multiple diversions within the cluster, the Diversion counter
parameter will be increase just by 1.
--]]

M = {}
function M.inbound_INVITE(msg)
    if msg:isReInviteRequest()
    then
        return
    end

    -- Get the Diversion header. If no Diversion header then return.
    local diversion = msg:getHeader("Diversion")
    if not diversion
    then
       return
    end
    local pt = msg:getPassThrough()
    pt:addHeader("X-Diversion", diversion)
end
function M.outbound_INVITE(msg)
    -- This script is applicable only for initial INVITE.
        if msg:isReInviteRequest()
        then
                return
        end
    -- Get Diversion header. If there is no Diversion header then return
        local diversion = msg:getHeader("Diversion")
        if not diversion
        then
                return
        end

        -- Get X-Diversion Header. If there was no X-Diversion header then return
        local xDiversion = msg:getHeader("X-Diversion")
        if not xDiversion
```

```
    then
            return
    end

-- Get URI from Diversion header
    local divString = msg:getUri("Diversion")
    if divString
    then

    -- Parse URI and get DN and Host
            local divuri = sipUtils.parseUri(divString)
            if divuri
            then
                    lrn_user = divuri:getUser()
                    lrn_host = divuri:getHost()

           end
    end

-- Get URI from X-Diversion header
    local xdivString = msg:getUri("X-Diversion")
    if xdivString
    then
       -- Parse URI and get DN and Host
            local xdivuri = sipUtils.parseUri(xdivString)
            if xdivuri
            then
                    xlrn_user = xdivuri:getUser()
                    xlrn_host = xdivuri:getHost()
            end
    end

-- Get counter parameter value from inbound diversion (X-Diversion).
local counter = msg:getHeaderValueParameter("X-Diversion", "counter")

-- If new LRN is different from incoming LRN, that means there was a local
    -- call forward on UCM, so increase the counter.
    if not ( string.match(lrn_user, xlrn_user) and string.match(lrn_host, xlrn_host) )
    then

    -- If there is no counter parameter, then find out how many Diversion
    -- headers are there. Set the counter to no. of Diversion Header.
    -- If there is a counter value, just increase it by 1.
            if not counter
            then

                   local xdiv = msg:getHeaderValues("X-Diversion")
                   if #xdiv > 1
                    then
                            counter = #xdiv
                    end
           else
                   counter = counter + 1
            end
            msg:addHeaderValueParameter("Diversion","counter", counter)
    else
           -- As there is no local call forwarding, so pass through the counter
           -- parameter.
            if counter
            then
       msg:addHeaderValueParameter("Diversion","counter", counter)
       end
    end
    msg:removeHeader("X-Diversion")
```

```
end
return M
```

メッセージの変換

クラスタ内でコール転送があり、着信 INVITE メッセージにカウンタ パラメータを持つ Diversion ヘッダーが含まれる場合、転送カウンタ スクリプトは、発信 INVITE メッセージのカウンタ パラメータを 1 だけ増やします。

元の着信メッセージ

```
INVITE sip:3002@10.10.10.53:5060 SIP/2.0^M
Via: SIP/2.0/TCP 10.10.10.51:5060;branch=z9hG4bK4b8a7eea6b41^M
From: <sip:1003@10.10.10.51>;tag=130169~d434b1d7-c1e3-44e7-a77e-abf211d2682e-26620367^M
To: <sip:3002@10.10.10.53>^M
Date: Fri, 04 Nov 2011 14:51:39 GMT^M
Call-ID: 7ef45500-eb31fbfb-3ec2-330a0a0a@10.10.10.51^M
Supported: timer,resource-priority,replaces^M
Min-SE:  1800^M
User-Agent: Cisco-CUCM8.6^M
Allow: INVITE, OPTIONS, INFO, BYE, CANCEL, ACK, PRACK, UPDATE, REFER, SUBSCRIBE, NOTIFY^M
CSeq: 101 INVITE^M
Expires: 180^M
Allow-Events: presence, kpml^M
Supported: X-cisco-srtp-fallback^M
Supported: Geolocation^M
Call-Info: <sip:10.10.10.51:5060>;method="NOTIFY;Event=telephone-event;Duration=500"^M
Cisco-Guid: 2129941760-0000065536-0000000148-0856295946^M
Session-Expires:  1800^M
Diversion: <sip:1001@10.10.10.51>;reason=no-
answer;privacy=off;screen=yes;counter=2,<sip:1006@10.10.10.51>;reason=no-
answer;privacy=off;screen=yes^M
P-Asserted-Identity: <sip:1003@10.10.10.51>^M
Remote-Party-ID: <sip:1003@10.10.10.51>;party=calling;screen=yes;privacy=off^M
Contact: <sip:1003@10.10.10.51:5060;transport=tcp>^M
Max-Forwards: 69^M
Content-Length: 0^
```

発信メッセージ

転送カウンタ スクリプトの実行後の発信メッセージに対する変更は、太字で示しています。

```
IINVITE sip:1005@10.10.10.51:5060 SIP/2.0^M
Via: SIP/2.0/TCP 10.10.10.53:5060;branch=z9hG4bK1d64062fa6f^M
From: <sip:1003@10.10.10.53>;tag=18448~94147210-61b5-4d32-a08d-5daf91ec321b-27003595^M
To: <sip:1005@10.10.10.51>^M
Date: Fri, 04 Nov 2011 14:51:44 GMT^M
Call-ID: 81ef4580-eb31fc00-e2-350a0a0a@10.10.10.53^M
Supported: timer,resource-priority,replaces^M
Min-SE:  1800^M
User-Agent: Cisco-CUCM8.6^M
Allow: INVITE, OPTIONS, INFO, BYE, CANCEL, ACK, PRACK, UPDATE, REFER, SUBSCRIBE, NOTIFY^M
CSeq: 101 INVITE^M
Expires: 180^M
Allow-Events: presence, kpml^M
Supported: X-cisco-srtp-fallback^M
Supported: Geolocation^M
Call-Info: <sip:10.10.10.53:5060>;method="NOTIFY;Event=telephone-event;Duration=500"^M
Cisco-Guid: 2179941760-0000065536-0000000121-0889850378^M
Session-Expires:  1800^M
Diversion: <sip:3002@10.10.10.53>;reason=no-
answer;privacy=off;screen=yes;counter=3,<sip:1006@10.10.10.51>;reason=no-
answer;privacy=off;screen=yes^M
```

```
P-Asserted-Identity: <sip:1003@10.10.10.53>^M
Remote-Party-ID: <sip:1003@10.10.10.53>;party=calling;screen=yes;privacy=off^M
Contact: <sip:1003@10.10.10.53:5060;transport=tcp>^M
Max-Forwards: 68^M
Content-Length: 0^M
```

CHAPTER 11

プリロードされたスクリプト

プリロードされたスクリプトは、Cisco Unified Communications Manager（Unified CM）基本インストールの一部として提供される Lua スクリプトです。プリロードされたスクリプトは、Unified CM Release 8.6 以降のリリースにアップグレード（またはインストール）すると自動的に使用できます。これらのスクリプトに、通常、特定の機能に使用でき、分かりやすい命名規則で名前付けされた過透性および正規化関連の Lua コードがあります。

管理者は、[SIP トランク正規化スクリプト（SIP Trunk Normalization Script）] ドロップダウン メニューでプリロードされたスクリプトを選択できます。プリロードされたスクリプトへの追加的な透過性または正規化関連の変更を追加するには、プリロードされたスクリプトの内容をコピーし、新しいスクリプトを、コピーされた内容と追加の必要な変更で作成します。

次のプリロードされるスクリプトが Unified CM Release 9.0 以降のリリースで提供されます。

- Refer-passthrough：SIP トランクである場合、コール アークの相手側への着信ダイアログ内参照 REFER（Replaces なし）メッセージを受け渡すために使用されます。Lua スクリプトでは、着信 REFER メッセージから Refer-To および Referred-By ヘッダーをコピーし、透過性オブジェクトに保存します。
- HCS-PCV-PAI-passthrough：このスクリプトは次の目的に使用されます。
  - INVITE、UPDATE、および 200 OK の P-Charging-Vector ヘッダーの受け渡し
  - P-Charging-Vector の 200 OK への追加時に設定された term-ioi を使用
  - INVITE の P-Asserted-Identity ヘッダーの受け渡し
- vcs-interop : Unified CM と VCS の間に適切な相互運用性を可能にするために使用します。このスクリプトは、2 つのノードがどのようにから SRTP をサポートするかの違いを特に扱い、From、Remote-Party-Id および P-Asserted-Id ヘッダーの URI の右側を変更して、Unified CM の IP アドレスの代わりに設定されたトップ レベル ドメインを使用します。
- 転送カウンタ：コールの転送および即転送シナリオの転送カウンタ パラメータを処理するために使用されます。このスクリプトは、着信および発信トランクの両方に接続する必要があります。クラスタ内でコールの即転送またはコールの転送がある場合、このスクリプトは発信 diversion ヘッダーのカウンタ パラメータを調整します。クラスタ内でコール転送が存在しない場合、このスクリプトは着信側から発信側にカウンタ パラメータを受け渡すだけです。転送カウンタ スクリプトの詳細については、「転送カウンタ」（P.10-7）を参照してください。

シスコシステムズ合同会社
〒107-6227 東京都港区赤坂9-7-1 ミッドタウン・タワー
http://www.cisco.com/jp
お問い合わせ先:シスコ コンタクトセンター
0120-092-255(フリーコール、携帯・PHS含む)
電話受付時間 : 平日 10:00〜12:00、13:00〜17:00
http://www.cisco.com/jp/go/contactcenter/