Cisco マクロ スクリプト作成のチュートリアル

Doc: D1539401

このチュートリアルの目的は、CE ビデオ システム用の最初のマクロ スクリプトを記述する方法を紹介することと、エディター内の例について詳しく説明することです。

JavaScript に関する知識は必要ありませんが、XAPI と Cisco のビデオ システムのある程度の経験は必要です。高度なマクロや製品品質のマクロを記述する場合は、より詳細な JavaScript コースを受講するか、他の方のアドバイスを求めることをお勧めします。

最大限の学習効果を得るには、本書を読みながらマクロ エディターを使用して、手動で例を入力し、それらに変更を加えて試してみることを推奨します。

XAPI の概要

XAPI の機能を簡単に要約します。

ステータスはビデオ システム内のプロパティであり、時間の経過と共に変化する性質です。リクエストで値を取得したり、変化したときに通知を受信したりできます。例：現在のシステム ボリューム、システムが通話中であるかどうか、他のユーザがプレゼンテーション中であるかどうか。

コマンドは、通話の発信、プレゼンテーションの開始、マイク音量の変更など、ビデオ システムを操作するための最も一般的な方法です。通常、これらのコマンドは一時的に変更され、リブート後には保持されません。通常、コマンドによってステータス値が変更されます。コマンドは、パラメータ（通話する相手やコール レートなど）を受け付けることができます。

設定は、システム名、デフォルトの音量、近接モード、壁紙の画像など、ビデオ システムのより永続的な設定です。これらの設定は、システムをリブートしても保持されます。

イベント リスナーを使用すると、イベント、ステータス値、または設定をリッスンできます。変更するたびに、新しい値で通知されます。

スタート ガイド

XAPI の使用を開始する前に、デバッグの方法を理解しておく必要があります。次のマクロを記述します。

```
console.log('Hello Macro');
```

保存して実行します。`Hello Macro` がコンソール ログに書き込まれていることを確認します。これは開発中に何が起こっているのか、コードのどの部分が実行されているのかなどを理解するのに役立ちます。また、ログが保存されるため、使用中に予期しない問題が発生した場合は、サポート チームが後で確認できます。

各 JavaScript のステートメントをセミコロンで終了することを忘れないでください。

XAPI ライブラリ オブジェクト

「XAPI」ライブラリを使用すると、マクロがビデオ システムに通信できます。インポートするには、次のように入力するだけです。

```
const xapi = require('xapi');
```

コード内の `xapi` オブジェクトを使用してコマンドを呼び出し、ステータス値を取得し、

次のすべての例では、最初にこのインポートを実行する必要があります。

コマンドの起動

TShell から XAPI を使用して通話するには、通常、次のようにします。

```
xCommand Dial Number: macro@polo.com
```

マクロからこれを実行するには、次のようにします。

```
xapi.command('Dial', { Number: 'macro@polo.com' });
```

保存して実行します。マクロは、開始後すぐにコールを行うはずです。次の注意事項があります。

- の後のドットは、`xapi` から関数を呼び出していることを意味します。`xapi` この例では次のものを呼び出しています：`command`
- `Dial` は関数の最初のパラメータであり、起動する xCommand ですinvoke

- 2 番目のパラメータはコマンドへのパラメータであり、この場合、コールする番号です
- 引数はオブジェクトであるため、次によって囲まれています：`{}`
- `Number` は引数オブジェクトのプロパティ名であり、アポストロフィを必要としません
- コマンド名とプロパティ値は文字列であり、"　で囲む必要があります

エラー処理

ライブラリを `xapi` 使用する際は、常にエラーをキャッチすることを推奨します。本書のほとんどの例では、読みやすさのために省略しています。

```
function handleError(error)
  { console.log('Error', error);
}

xapi.command('Audio Incorrect Command').catch(handleError);
```

通常、JavaScript では、下から上へとコードを読む必要があります。関数と変数は、使用する前に定義します。

複数の引数

XAPI を使用してデュアル モニタのコンテンツを切り替える XAPI コマンドは次のようになります。

```
xCommand Video Matrix Swap OutputA: 1 OutputB: 2
```

マクロを使用してこれを行うには、次のようにします。

```
xapi.command('Video Matrix Swap',
  { OutputA: '1',
  OutputB: '2',
});
```

オブジェクト プロパティをコンマで区切しているため、読みやすくするために別々の行に配置することができます。

イベント

前の例では、マクロが開始されるとすぐにコマンドが実行されました。これは、通常、ビデオ システムが起動し終わった場合です。これでは少々不便です。マクロは、特定のイベントをリッスンし、それらのイベントが発生したときにアクションを実行するのがより一般的です。たとえば、カスタムのクイック ダイヤル パネルで室内にあるコントロール ボタンが押されると番号を呼び出すことです。

XAPI の室内のコントロール ボタンをリッスンするには、次のようにします。

```
xFeedback Register Event/UserInterface/Extensions/Widget/Action
```

ボタンをクリックすると、通常、次のようなイベントが表示されます。

```
*e UserInterface Extensions Widget Action WidgetId: 'quickdial'
*e UserInterface Extensions Widget Action Type: 'clicked'
*e UserInterface Extensions Widget Action Value: ''
```

特定のウィジェットをクリックしたら番号をダイヤルするマクロを作成するには、次のようにします。

```
function quickDial(event) {
  if (event.WidgetId === 'quickdial' && event.Type === 'clicked')
    { xapi.command('Dial', { Number: 'macro-polo@cisco.com' });
  }
}

xapi.event.on('UserInterface Extensions Widget Action', quickDial);
```

- 最初に、コールを設定するための関数を定義します（この例では `quickDial` と呼びます）
- 次に、室内コントロールのイベントとこの機能を接続します。これ `xapi.event.on` には、次を使用します：
- イベント ハンドラーは、マクロが実行されている限り、リッスンを続けます
- 2 番目のパラメータ `quickDial` は、室内コントロールイベントが発生した際に常にコールできるようにする関数の名前です
- 他のタイプの室内コントロール イベントが発生したときにコールが開始されないように、アクション タイプとウィジェット ID を確認します
- `===` は、値が完全一致する必要があることを意味しています。JavaScript は独特です。

イベント リスナー`quickDial` に関数の名前を通知するために を送信します。これは、JavaScript の**コールバック**として知られています。代わりに `quickDial()` を入力すると、意図したとおりに機能しません。

フィードバック リスナーの停止:

イベント リスナーを停止するには、次のようにします。

```
const stop = xapi.event.on('UserInterface Extensions Widget Action', quickDial);

// ... ここに他のコードを挿入

stop(); // イベントへのリッスンを停止
```

// ここでの は、その行の残りがコメントであることを意味し、マクロでは無視されます。これらはコードが実行する内容を説明するために使用されます。

ステータスのフィードバック

イベントは、ボタンの押下などの 1 つのポイントで発生します。また、イベントと同様の方法で、xStatus の変更に関する通知を受けることもできます。XAPI でシステム ボリュームが変更されたときに通知されるようにするには、次のようにします。

```
xFeedback Register Status/Audio/Volume/Level
```

`const` キーワードは定数を定義します。`const` マクロが開始されると、これは変更できません。これは、マクロの先頭で定義しておくことを推奨します。これにより、簡単に見つけて変更できます。値を複数回入力する場合は、常に、値を に置き換えることを検討する必要があります。

マクロで値を 80% に制限するには、次のようにします。

```
const MAX_VOLUME = 80;

function onVolumeChange(volume)
  { if (volume > MAX_VOLUME) {
    xapi.command('Audio Volume', { Level: MAX_VOLUME });
  }
}

xapi.status.on('Audio Volume Level', onVolumeChange);
```

マクロが有効の間に変数の値を変更する必要がある場合は、`let` を次の代わりに使用します：`const` 。

`const` はハードコーディングされた値であり、一方 `let` は動的に変更できます。

ステータス リクエスト

ステータス値を常にリッスンするのではなく、直接要求する必要がある場合があります。これには`xapi.status.get` 関数を使用します。

先述のクイック ダイヤル ボタンが押されたときに、すでに通話中であるかどうかを確認するには、次のようにします。

```
function callIfNotInCall(callCount) {
  if (callCount < 1) {
    xapi.command('Dial', { Number: 'macro@polo.com' });
    console.log('Call Macro Polo');
  }
}

function onInroomEvent(event)
  { console.log('In-room event occured',
  event);
  if (event.WidgetId === 'quickdial' && event.Type === 'clicked')
    { xapi.status.get('SystemUnit State NumberOfActiveCalls')
    .then(callIfNotInCall);
  }
}

xapi.event.on('UserInterface Extensions Widget Action', onInroomEvent);
```

`then` 関数は、少々特殊です。ビデオ システムに通話の番号を要求すると、この要求には少 々時間がかかります。プロセッサをブロックするのではなく、オブジェクト（**プロミス**と呼ばれます）を返します。`callIfNotInCall` これは、応答を取得した後すぐに別の関数を回答 （通話の番号）とともに呼び出します。

ログ出力を見ると、室内のイベント ログは、コード内の順序と逆に表示されますが、通話のログの前にあることに注意してください。これは、プログラムの実行が継続し、クエリの結果が数ミリ秒後に返されるためです。

このことが厄介と感じられる場合は、その構文を理解するようにしてみてください。これは応答性の高いプログラミングを実行するための共通の JavaScript パターンであるためです。

詳細については以下を参照してください。

ダミーのプロミス

コードを記述するための別の方法（構文）

ここまでの例では、すべての関数に名前を付けています。エディターの例にあるように、名前を付けずに、**矢印関数**（ `=>` ）を使用した関数もあります。

例：

```
function printVolume(volume)
  { console.log('Volume is', volume);
}

xapi.status.on('Audio Volume', printVolume);
```

これは、次のように記述することもできます。

```
xapi.status.on('Audio Volume', (volume) => console.log('Volume is', volume));
```

これが紛らわしいと感じる場合は、よりわかりやすい構文を使います。ただし、構文が他のコードにある場合は、構文を把握することをお勧めします。

高度：プロミスをチェーンする

アクションを実行する前に、他のいくつかの操作を実行する必要がある場合があります。たとえば、システムをスリープ状態にする前に、自分が通話に参加しているかどうか、またはルームにユーザがいるかどう（ユーザのカウント）などを確認する場合です。

これを行う 1 つの方法は、次のようにプロミスを「チェーン」することです。

```
function checkPeopleCount(people) {
  if (people < 1) xapi.command('Standby Activate');
}

function checkInCall(calls) {
  if (calls < 1) xapi.status.get('RoomAnalytics PeopleCount Current')
  .then(checkPeopleCount)
  .catch(console.error);
};

xapi.status.get('SystemUnit State NumberOfActiveCalls')
.then(checkInCall)
.catch(console.error);
```

さらに、次のように、よりコンパクトで並列な方法でこれを行うこともできます。

```
const p1 = xapi.status.get('RoomAnalytics PeopleCount Current', checkPeopleCount)
.catch(console.log);
const p2 = xapi.status.get('SystemUnit State NumberOfActiveCalls')
.catch(console.log);

Promise.all([p1, p2]).then(results => {
  const peopleCount = results[0];
  const callCount = results[1];
  if (peopleCount < 1 && callCount < 1) xapi.command('Standby Activate');
});
```

`Promise.all` は、両方の結果が利用可能になった後も継続され、両方の結果が `then` に配列で送信されます。2 回目の要求を実行する前に最初の結果を待たないため、これも少し高速です。繰り返しになりますが、この構文に慣れるには時間がかかる場合があります。試してみてください。

構成

`xapi.config.set` と `xapi.config.get` を使用して、構成を取得および設定します。たとえば、自動応答がオンになっているかどうかを確認するには、次のようにします。

```
xapi.config.get('Conference AutoAnswer Mode', v => console.log('Mode', v));
```

構成の場合は、1 つしか存在できないため、パラメータをオブジェクトとして指定する必要はありません。マイク 2 のレベルを調整するには、次のようにします。

```
xapi.config.set('Audio Input Microphone 2 Level', 33);
```

構成の通知は、ステータスとイベント通知に似ています。近接モードが切り替えられたときに印刷するには、次のようにします。

```
xapi.config.on('Proximity Mode', v => console.log('Proximity changed to ', v));
```

オブジェクトの操作

前述の例のほとんどは、音量レベル、コール数などの単数値を使用しています。

また、オブジェクトを `xapi` から戻したり、オブジェクトのプロパティにドット `.` アクセスしたりできます。

```
xapi.status.get('Conference')
.then(conf => console.log(conf.DoNotDisturb, conf.Presentation.Mode));
```

オブジェクトのすべてのプロパティを表示する場合は、`console.log` を次のように使用します。

```
xapi.status.get('Video Input Connector', c => console.log(c));
// => { id: "2", Connected: "False", SignalState: "Unknown", Type: "DVI" }
```

リストの使用

XAPI には、要素のリストを含めることもできます。タイプ「PC」の入力ソースを見つけて、そのプレゼンテーションを開始するには、次のようにします。

```
function present(connector) {
  xapi.command('Presentation Start', { ConnectorId: connector });
}

xapi.config.get('Video Input Connector')
.then(list => {
  for (let i = 0; i < list.length; i++) {
    const con = list[i];
    console.log(`Connector ${con.id}: ${con.Type}`);
    if (con.Type === 'PC') present(con.id);
  }
});
```

ここで、いくつか注意点があります。

1. 今回は、構成クエリの結果はリストであり、単数値ではありません
2. 上記の構文を使用して配列をループできます
3. 上記の `` `Connector ${i}` `` の特殊な部分により、変数を文字列に挿入することができます
4. JavaScript 配列はインデックス 0 で開始されますが、XAPI リストは 1 で始まるため、`.id` の代わりに、リストの ^{i} プロパティを使用して、正しいコネクタを取得します

タイマー

例からわかるように、マクロをプログラムするには、イベントと更新に対応させることをお勧めします。もう 1 つの方法は、システムを定期的にポーリングする方法です。たとえば、通話中であるかどうかを 10 秒ごとに問い合わせます。これは、応答性が低く、必要なリソースが多く、通常エラーの発生しやすい方法であるため、あまり推奨されません。

それでも、タイマーを使用する理由はあります。このために、通常の JavaScript メソッドがサポートされており、`setTimeout` 1 つのタイマーを開始するために、`setInterval` は定期的なタイマーのために、そして対応する `clearTimeout` はそれらを停止するために使用します。`clearInterval`

たとえば、1 時間おきにシステムのミュートを解除するには、次のようにします。

```
function unmute() {
  xapi.command('Audio Microphones Unmute');
}

const everyHour = 60 * 60 * 1000; // ミリ秒単位で
const timerId = setInterval(unmute, everyHour);

// タイマーを停止するには：
clearInterval(timerId);
```

コールバック `ミュート` (`setInterval` 。

スケジューリング

JavaScript やマクロ フレームワークには、特定の時間帯にアクションをスケジュールするための標準的な方法はありません。ただし、この関数をマクロにコピーして使用することができます。

```
function schedule(time, action) {
  let [alarmH, alarmM] = time.split(':');
  let now = new Date();
  now = now.getHours() * 3600 + now.getMinutes() * 60 + now.getSeconds();
  let difference = parseInt(alarmH) * 3600 + parseInt(alarmM) * 60 - now;
  if (difference <= 0) difference += 24 * 3600;
  return setTimeout(action, difference * 1000);
}
```

次に、たとえば、すべての平日で 09:00 に会議をダイヤルするには次のようにします。

```
const StandupTime = '09:00';
const StandupUri = 'macro@polo.com';
const Sunday = 0, Saturday = 6;
```

```
function dialStandup() {
  const weekDay = new Date().getDay();
  if (weekDay !== Sunday && weekDay !== Saturday)
    { xapi.command('Dial', { Number: StandupUri });
  }

  schedule(StandupTime, dialStandup); // 翌日にスケジュール
}

schedule(StandupTime, StandupUri);
```

今日のコール時に明日の会議がスケジュールされており、これが繰り返されます。

パフォーマンスとライフ サイクル

マクロ システムには安全メカニズムが組み込まれているため、マクロによってパフォーマンス上の問題が発生することはありません。マクロの**ランタイム** プロセスは、定期的に測定されます。マクロまたはシステムの一般に大きな負荷がかかっている場合は、ランタイムを一時的に停止して、ビデオ システムが継続して動作するようにできます。スマートフォンのサード パーティ製アプリと同様に、OS は警告なしで攻撃的なマクロやバグのあるマクロを停止する場合があります。しばらくすると、ランタイムが自動的にすべての有効なマクロを再起動します。

このため、次のことを推奨します。

- マクロにステータスを含めないようにし、代わりにシステムからステータスを検出してみてください。たとえば、システムが通話中であるかどうかを記憶するための変数を使用する代わりに、マクロの起動時にシステムにこれを問い合わせます。
- 同時にあまり多くのことを実行しないでください。たとえば、システムが通話に入るときは多くのリソースが使用されます。そのため、大量の XAPI 要求を行う必要があり、マクロが終了されそうな場合は、`setTimeout` を使用して少し分散するようにします。
- 実行させる実際のビデオシステム上と、似た環境やより劣悪な環境でマクロをテストします。プレゼンテーションなどの複数のサイト コールでテストして、すべてがスムーズに実行されることを確認します。

ユーザ インターフェイス要素

マクロには、ユーザが選択できるようにしたり、ユーザに情報を与えたりするために、ユーザ インターフェイス要素が必要な場合があります。ここでは、提供されている最も一般的なコンポーネントとその使用方法の例を紹介します。完全なリファレンスについては、CE API ガイド全体を常に参照してください。

アラート

アラートは、ビデオ画面とタッチ画面の両方の中央に、選択したテキストとともにポップアップされます。ユーザはタッチしてこれを閉じたり、タイムアウト後に自動的に閉じさせたりすることができます。

```
xapi.command('UserInterface Message Alert Display',
  { Title: 'Volume limiter',
  Text: 'カスタムのスクリプトによりボリュームをこれ以上大きく設定できなくなっています',
  Duration: 10,
});
```

通知

通知はアラートと非常に似ていますが、目立つようになっておらず、メイン ウィンドウ画面と画面の右上隅にしか表示されません。

```
xapi.command('UserInterface Message TextLine Display',
  { Text: '残り時間 1 分',
  Duration: 1,
});
```

後から他のマクロ（またはビデオ システム）によってアラートまたは通知が表示された場合、そのマクロによって現在のものは削除されるので、テキストを永続的に表示することはできないことに注意してください。

プロンプト

プロンプトはポップアップされ、ユーザは事前に定義された選択肢のリストから選択することができます。ユーザが選択すると、対応可能なイベントが生成されます。`FeedbackId` を使用すると、どのプロンプトが応答を生成したかを識別できます（マクロに複数のダイアログを含めることができます）。`event.OptionId` は、ユーザが選択した実際の選択を示します。

通話の終了後、通話品質アンケートを表示します。

```
function showSurvey() {
  xapi.command('UserInterface Message Prompt Display', {
    Title: '通話品質アンケート',
```

```
    Text: 'この通話の品質はどうでしたか？',
    FeedbackId: '通話品質', 'Option.1':'良い',
    'Option.2':'まぁまぁ',
    'Option.3': '悪い'
  });
}

xapi.event.on('UserInterface§ Message Prompt Response', (event) =>
  { if (event.FeedbackId !== 'call-quality') return;
  console.log('Quality:', event.OptionId);
});

xapi.event.on('CallDisconnect', (event) => {
    if(event.Duration > 10) showSurvey();
});

showSurvey();
```

現時点では、マクロはこの情報をどこにも保存または送信できませんが、今後のリリースでは変更される可能性があります。

文字入力

プロンプトと似ていますが、定義済みの選択肢ではなく、ユーザがテキストを入力できるようにします。

以下により、ユーザはビデオのエンドポイントの名前を入力できます。

```
xapi.event.on('UserInterface Message TextInput Response', (event) => {
  if (event.FeedbackId === 'system-name')
    { xapi.config.set('SystemUnit Name', event.Text);
  }
});

xapi.command('UserInterface Message TextInput Display',
  { FeedbackId: 'system-name',
  Title: 'システム名を選択してください',
  Text: '他のユーザがコールすると、この名前が表示されます',
});
```

室内制御

室内制御では、トグル、スライダー、ボタン、ラジオ ボタンなどのウィジェットを含む完全にカスタマイズ可能なユーザ インターフェイスを利用できます。他のビデオ システムのコンポーネントと同じように、XAPI を介してマクロと通信します。

すべての室内制御ウィジェットには、固有の**ウィジェット ID** が割り当てられています。ボタンが押された、またはスイッチが切り替えられたなどのアクションが発生すると、XAPI イベントが生成され、マクロはこれをリッスンして応答できます。

次の例では、物理的な音量ボタンを押すのではなく、音量をスライダーで調整できます。

```
function onUiAction(event) {
  if (event.WidgetId !== 'volume_slider') return;
  const newVolume = parseInt(event.Value * 100 / 255);
  xapi.command('音声ボリューム設定', { Level: newVolume });
}

function syncUi(level) {
  xapi.command('UserInterface Extensions Widget SetValue', {
    WidgetId: 'volume_slider',
    Value: level * 255 / 100,
  });
}

// UI イベントをリッスン
xapi.event.on('UserInterface Extensions Widget Action', onUiAction);

// UI を同期
xapi.status.on('Audio Volume', syncUi);

// 初期値を修正
xapi.status.get('Audio Volume', syncUi);
```

マクロ エディターの使用例の項で、他の例を確認してください。

注：室内制御は完全な双方向の API であることが想定されています。これは、超音波のペアリングを変更するためのマクロや室内パネルを作成して、他のユーザが他の場所（別の Touch 10、ウェブ管理者ページ、コマンド ラインなど）から設定を変更した場合、UI が同期されていることを確認する必要があることを意味します。つまり、常にステータスの変更（`ウィジ SetValue`）を確認する必要があります。元の GUI が正しい状態を示している場合もです。

室内制御に関する詳細なリファレンス ガイドについては、室内制御エディターにアクセスしてください。

ヒント

- マクロが反応するようにするには、ポーリングではなくイベントやフィードバックを使用します。
- マクロは、システムの通常の動作を完全に変更することができます。マクロにより通常のユーザが驚いたり混乱する可能性がある場合は、ビデオ システムの画面に通知を表示するなどして 知らせるようにします。
- 他のマクロに依存したマクロを作成することはお勧めできません。もちろん、複数のマクロのアクションが相互に依存しない限り、呼び出し状態など同じ XAPI 値をリッスンする複数のマクロを作成することはできます。マクロが実行される順序は保証されません。
- 現在マクロで、部屋の照明を制御するなど、外部からデータを取得したり、外部に送信したりすることはできません。これを行うには、外部制御システムが必要になります。その場合でも、マクロと外部システム（低レベルの照明制御用の Crestron など）を組み合わせ、プレゼンテーションや通話のステータスに応じてマクロが照明を調整できるようにすることはできます。
- マクロが大きすぎたり複雑すぎるようになったら、代わりにマクロを外部インテグレーションにすることを検討してください。たとえば、同じコードを調整して、たとえば JavaScript を実行しているスタンドアロンの Node サーバ上で実行するようにできます。

失敗した場合

マクロ処理では、異常が発生するとマクロが自動的に停止されます。マクロのフレームワークの再起動中にマクロを無効にできます。マクロを完全に無効にするには、次のキル スイッチを使用します：`xConfig Macros Mode: Off`。

大失敗した場合

マクロの起動時にブートするマクロを作成した場合、少々複雑です。マクロが再起動されると同じ問題が再度発生するからです。

ブート中、マクロのプロセスが開始される前にマクロを無効にするには、コマンド ラインで次のように操作します。

```
while sleep 1
do
  echo "xConfig Macros AutoStart: Off" | ssh root@your-ip "/bin/tsh"
done
```

これで、エラーのあるマクロを無効にしたり修正したりして、マクロのランタイムを再起動できます。問題が解決したら、自動起動を再度設定するようにしてください。

以上です

自分だけのマクロを有意義な時間をお過ごしください。作成したマクロが他のユーザにとって便利であると思われる場合は、私達にぜひ共有してください。