



EEM CLI ライブラリのコマンド拡張

すべてのコマンドラインインターフェイス (CLI) ライブラリ コマンド拡張は、`::cisco::eem` 名前空間に属します。

このライブラリによって、ユーザーに対し、CLI コマンドを実行し、Tel でコマンドの出力を取得する機能が用意されます。コマンドが `exec` によって実行され、コマンドの出力が読み戻されるようにするため、ユーザーは、このライブラリでコマンドを使用して、`exec` を生成し、それに対して仮想端末チャンネルをオープンし、コマンドを記述してチャンネルに対して実行できます。

CLI コマンドには、対話式コマンドと非対話式コマンドの、2つのタイプがあります。

対話式コマンドでは、コマンドの入力後、デバイスによって異なるユーザーオプションが質問される「Q&A」フェーズがあり、ユーザーは、各質問に対する答えを入力する必要があります。すべての質問が適切に答えられた後、ユーザーのオプションに従って、完了するまでコマンドが実行されます。

非対話式コマンドでは、コマンドが一度入力されると、コマンドが完了まで実行されます。EEM スクリプトを使用してさまざまなタイプのコマンドを実行するには、異なる CLI ライブラリ コマンドシーケンスを使用する必要があります。詳細については、`cli_write Tel` コマンドの「CLI ライブラリを使用した非対話式コマンドの実行」の項および「CLI ライブラリを使用した対話式コマンドの実行」の項を参照してください。

`vty` 行は、`line vty` CLI コンフィギュレーション コマンドを使用して設定された `vty` 行のプールから割り当てられます。EEM によって `vty` 行が使用されていない場合で、使用可能な `vty` 行がある場合、EEM では、`vty` 行が使用されます。EEM によって `vty` 行がすでに使用されている場合で、使用可能な 3 行以上の `vty` 行がある場合も、EEM では、`vty` 行が使用されます。3 行よりも少ない `vty` 行が使用可能な場合、残りの `vty` 行は Telnet で使用するために予約されているので、接続は失敗することに注意してください。

お使いのリリースで XML-PI がサポートされている場合があります。XML-PI サポート、新しい CLI ライブラリ コマンド拡張、および、XML-PI の実装方法の例については、「EEM CLI ライブラリ XML-PI サポート」を参照してください。

- [cli_close](#) (2 ページ)
- [cli_exec](#) (2 ページ)
- [cli_get_ttyname](#) (3 ページ)

- [cli_open](#) (3 ページ)
- [cli_read](#) (4 ページ)
- [cli_read_drain](#) (5 ページ)
- [cli_read_line](#) (6 ページ)
- [cli_read_pattern](#) (6 ページ)
- [cli_run](#) (7 ページ)
- [cli_run_interactive](#) (8 ページ)
- [cli_write](#) (9 ページ)
- [EEM 4.0 CLI ライブラリ XML-PI サポート](#) (12 ページ)
- [EEM CLI ライブラリ XML-PI サポート](#) (12 ページ)

cli_close

exec プロセスをクローズし、コマンドラインインターフェイス (CLI) に接続された、vty および指定されたチャンネルハンドラをリリースします。

構文

```
cli_close fd tty_id
```

引数

fd	(必須) CLI チャンネルハンドラ。
tty_id	(必須) cli_open コマンド拡張から返された TTY ID。

結果文字列

なし

_cerrno を設定

チャンネルをクローズできない。

cli_exec

指定されたチャンネルハンドラにコマンドを記述し、コマンドを実行します。次に、チャンネルからコマンドの出力を読み取り、出力を返します。

構文

```
cli_exec fd cmd
```

引数

fd	(必須) コマンドラインインターフェイス (CLI) チャンネルハンドラ。
cmd	(必須) 実行する CLI コマンド。

結果文字列

実行された CLI コマンドの出力。

_cerno を設定

チャンネルを読み取れない。

cli_get_ttyname

該当する TTY ID の実際と疑似の TTY の名前を返します。

構文

```
cli_get_ttyname tty_id
```

引数

tty_id	(必須) cli_open コマンド拡張から返された TTY ID。
--------	---

結果文字列

```
pty %s tty %s
```

_cerno を設定

なし

cli_open

vty を割り当て、EXEC コマンドラインインターフェイス (CLI) セッションを作成し、vty をチャンネルハンドラに接続します。チャンネルハンドラを含む配列を返します。



(注) **cli_open** への各コールによって、Cisco IOS vty 回線を割り当てる Cisco IOS EXEC セッションが開始されます。vty は、**cli_close** ルーチンが呼び出されるまで、使用中のままです。vty 行は、**line vty CLI** コンフィギュレーション コマンドを使用して設定された vty 行のプールから割り当てられます。EEM によって vty 行が使用されていない場合で、使用可能な vty 行がある場合、EEM では、vty 行が使用されます。EEM によって vty 行がすでに使用されている場合で、使用可能な 3 行以上の vty 行がある場合も、EEM では、vty 行が使用されます。3 行よりも少ない vty 行が使用可能な場合、残りの vty 行は Telnet で使用するために予約されているので、接続は失敗することに注意してください。

構文

```
cli_open
```

引数

なし

結果文字列

```
"tty_id {%s} pty {%d} tty {%d} fd {%d}"
```

イベントタイプ	説明
tty_id	TTY ID。
pty	PTY デバイス名。
tty	TTY デバイス名。
fd	CLI チャネルハンドラ。

_cerno を設定

- EXEC の pty を取得できない。
- EXEC CLI セッションを作成できない。
- 最初のプロンプトを読み取れない。

cli_read

読み取られている内容でデバイスプロンプトのパターンが発生するまで、指定されたコマンドラインインターフェイス (CLI) のチャネルハンドラからコマンド出力を読み取ります。一致するまで、読み取られたすべての内容を返します。

構文

```
cli_read fd
```

引数

fd	(必須) CLI チャンネルハンドラ。
----	---------------------

結果文字列

読み取られたすべての内容。

_cernno を設定

デバイス名を取得できない。



(注) この Tcl コマンド拡張によって、デバイス プロンプトを待つ状態がブロックされ、読み取られた内容が表示されます。

cli_read_drain

指定されたコマンドラインインターフェイス (CLI) のチャンネルハンドラのコマンド出力を読み取り、排出します。読み取られたすべての内容を返します。

構文

```
cli_read_drain fd
```

引数

fd	(必須) CLI チャンネルハンドラ。
----	---------------------

結果文字列

読み取られたすべての内容。

_cernno を設定

なし

cli_read_line

指定されたコマンドラインインターフェイス（CLI）のチャンネルハンドラから、コマンド出力の 1 行を読み取ります。読み取られた回線を返します。

構文

```
cli_read_line fd
```

引数

fd	(必須) CLI チャンネルハンドラ。
----	---------------------

結果文字列

読み取られた回線。

_cerno を設定

なし



(注) この Tcl コマンド拡張によって、行の末尾を待つ状態がブロックされ、読み取られた内容が表示されます。

cli_read_pattern

読み取られている内容でパターンが発生するまで、指定されたコマンドラインインターフェイス（CLI）のチャンネルハンドラからコマンド出力を読み取ります。一致するまで、読み取られたすべての内容を返します。



(注) パターンマッチロジックで、Cisco IOS コマンドから配信されるコマンド出力データを探すことによって、照会が試行されます。照会は、出力バッファの最新の 256 文字で常に行われます。ただし、使用可能な文字がより少ない場合は、より少ない文字で照会が行われます。正常な一致に 256 よりも多い文字が必要な場合、パターンマッチは実行されません。

構文

```
cli_read_pattern fd ptn
```

引数

fd	(必須) CLI チャンネルハンドラ。
ptn	(必須) チャンネルからコマンド出力を読み取る際に、パターンが照会されます。

結果文字列

読み取られたすべての内容。

_cerno を設定

なし



-
- (注) この Tcl コマンド拡張によって、指定されたパターンを待つ状態がブロックされ、読み取られた内容が表示されます。
-

cli_run

clist にある回数を繰り返し、それぞれが、イネーブルモードで実行されるコマンドラインインターフェイス (CLI) であることを前提とします。正常に実行されると、実行されたすべてのコマンドの出力を返します。失敗すると、失敗からのエラーを返します。

構文

```
cli_run clist
```

引数

clist	(必須) 実行されるコマンドのリスト。
-------	---------------------

結果文字列

出力されるすべてのコマンドの出力、またはエラーメッセージ。

_cerno を設定

なし。

使用例

次に、cli_run コマンド拡張の使用例を示します。

```
set clist [list {sh run} {sh ver} {sh event man pol reg}]
cli_run { clist }
```

cli_run_interactive

3つの項目がある `clist` のサブリストを提供します。正常に実行されると、実行されたすべてのコマンドの出力を返します。失敗すると、失敗からのエラーを返します。可能な場合には、配列も使用します。予測と応答を別々に保持することによって、より簡単に後で読み取ることができます。

構文

```
cli_run_interactive clist
```

引数

clist	<p>(必須) 3つの項目のリスト:</p> <ul style="list-style-type: none"> • command: 実行するコマンド • expect: 予想される応答プロンプトの正規表現パターンマッチ • responses: 2つの項目の配列として構成された応答プロンプトに対して可能性がある応答のリスト <ul style="list-style-type: none"> • expect: 可能性がある応答プロンプトの正規表現パターンマッチ • reply: その予測されるプロンプトの応答
-------	--

結果文字列

出力されるすべてのコマンドの出力、またはエラーメッセージ。各コマンドが実行されると、その出力が結果の変数に追加されます。入力リストが枯渇すると、CLIチャンネルが閉じ、集約結果が返されます。

`_cerrno` を設定

なし。

使用例

次に、`cli_run_interactive` コマンド拡張を使用してインターフェイス `fa0/0` のカウンタをクリアする例を示します。

```
set cmdarr(command) "clear counters fa0/0"
set cmdarr(responses) [list]
set resps(expect) {[confirm]}
set resps(reply) "y"
lappend cmdarr(responses) [array get resps]
set rc [catch {cli_run_interactive [list [array get cmdarr]]} result]
```

発生する可能性があるエラーには、次のようなものがあります。

- `exec` の `pty` を取得できない。
- `exec` を生成できない。
- 最初のプロンプトを読み取れない。
- チャネルを読み取れない。
- チャネルをクローズできない。

cli_write

指定された CLI チャネルハンドラに対して実行されるコマンドを書き込みます。CLI チャネルハンドラによって、コマンドが実行されます。

構文

```
cli_write fd cmd
```

引数

fd	(必須) CLI チャネルハンドラ。
cmd	(必須) 実行する CLI コマンド。

結果文字列

なし

`_cernno` を設定

なし

使用例

たとえば、次のように、コンフィギュレーション CLI コマンドを使用して、イーサネットインターフェイス 1/0 をアップにします。

```
if [catch {cli_open} result] {
  puts stderr $result
  exit 1
} else {
  array set clil $result
}
if [catch {cli_exec $clil(fd) "en"} result] {
  puts stderr $result
  exit 1
}
if [catch {cli_exec $clil(fd) "config t"} result] {
  puts stderr $result
  exit 1
}
```

```

}
if [catch {cli_exec $cli1(fd) "interface Ethernet1/0"} result] {
puts stderr $result
exit 1
}
if [catch {cli_exec $cli1(fd) "no shut"} result] {
puts stderr $result
exit 1
}
if [catch {cli_exec $cli1(fd) "end"} result] {
puts stderr $result
exit 1
}
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
puts stderr $result
exit 1
}

```

CLI ライブラリを使用した非対話式コマンドの実行

非対話式コマンドを実行するには、**cli_exec** コマンド拡張を使用して、コマンドを発行し、次に、出力とデバイスプロンプトを待ちます。たとえば、コンフィギュレーションCLIコマンドを使用して、イーサネット インターフェイス 1/0 をアップにする例を示します。

```

if [catch {cli_open} result] {
error $result $errorInfo
} else {
set fd $result
}
if [catch {cli_exec $fd "en"} result] {
error $result $errorInfo
}
if [catch {cli_exec $fd "config t"} result] {
error $result $errorInfo
}
if [catch {cli_exec $fd "interface Ethernet1/0"} result] {
error $result $errorInfo
}
if [catch {cli_exec $fd "no shut"} result] {
error $result $errorInfo
}
if [catch {cli_exec $fd "end"} result] {
error $result $errorInfo
}
if [catch {cli_close $fd} result] {
error $result $errorInfo
}
}

```

CLI ライブラリを使用した対話式コマンドの実行

対話式コマンドを実行するには、次の3つのフェーズが必要です。

- フェーズ1: **cli_write** コマンド拡張を使用して、コマンドを発行します。
- フェーズ2: Q&A フェーズ。**cli_read_pattern** コマンド拡張を使用して質問を読み取り（質問テキストの照合に指定される通常パターン）、**cli_write** コマンド拡張を使用して、代わりに回答を書き戻します。

- フェーズ 3：非対話式フェーズ。すべての質問が回答され、完了までコマンドが実行されます。**cli_read** コマンド拡張を使用して、コマンドの出力とデバイスプロンプトを待ちます。

たとえば、CLI コマンドを使用して、ブートフラッシュをまとめます。Tel 変数 `cmd_output` に、このコマンドの出力を保存します。

```

if [catch {cli_open} result] {
error $result $errorInfo
} else {
array set cli1 $result
}
if [catch {cli_exec $cli1(fd) "en"} result] {
error $result $errorInfo
}

# Phase 1: issue the command
if [catch {cli_write $cli1(fd) "squeeze bootflash:"} result] {
error $result $errorInfo
}

# Phase 2: Q&A phase
# wait for prompted question:
# All deleted files will be removed. Continue? [confirm]
if [catch {cli_read_pattern $cli1(fd) "All deleted"} result] {
error $result $errorInfo
}
# write a newline character
if [catch {cli_write $cli1(fd) "\n"} result] {
error $result $errorInfo
}
# wait for prompted question:
# Squeeze operation may take a while. Continue? [confirm]
if [catch {cli_read_pattern $cli1(fd) "Squeeze operation"} result] {
error $result $errorInfo
}
# write a newline character
if [catch {cli_write $cli1(fd) "\n"} result] {
error $result $errorInfo
}

# Phase 3: noninteractive phase
# wait for command to complete and the router prompt
if [catch {cli_read $cli1(fd) } result] {
error $result $errorInfo
} else {
set cmd_output $result
}
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
error $result $errorInfo
}

```

次に、CLI **reload** コマンドを使用して、デバイスがリロードされる例を示します。EEM **action_reload** コマンドによって、より効率的な方法で同じ結果が達成されますが、この例は、対話式コマンド実行での CLI ライブラリでの柔軟性を示すために示します。

```

# 1. execute the reload command
if [catch {cli_open} result] {
error $result $errorInfo
} else {

```

```

        array set cli1 $result
    }
    if [catch {cli_exec $cli1(fd) "en"} result] {
        error $result $errorMsg
    }
    if [catch {cli_write $cli1(fd) "reload"} result] {
        error $result $errorMsg
    } else {
        set cmd_output $result
    }
    if [catch {cli_read_pattern $cli1(fd) ".*(System configuration has been modified. Save\\|?
    \\|\\|yes/no\\|\\|: )"} result] {
        error $result $errorMsg
    } else {
        set cmd_output $result
    }
    if [catch {cli_write $cli1(fd) "no"} result] {
        error $result $errorMsg
    } else {
        set cmd_output $result
    }
    if [catch {cli_read_pattern $cli1(fd) ".*(Proceed with reload\\|\\|? \\|\\|confirm\\|\\|)"}
    result] {
        error $result $errorMsg
    } else {
        set cmd_output $result
    }
    if [catch {cli_write $cli1(fd) "y"} result] {
        error $result $errorMsg
    } else {
        set cmd_output $result
    }
    if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
        error $result $errorMsg
    }
}

```

EEM 4.0 CLI ライブラリ XML-PI サポート

EEM CLI ライブラリ XML-PI サポート

XML プログラマチック インターフェイス (XML-PI) が Cisco IOS Release 12.4(22)T で導入されました。XML-PI は異なるシスコ製品間で矛盾のない方法で、IOS コマンドライン インターフェイス (CLI) `show` コマンドを XML 形式にカプセル化した、プログラム可能なインターフェイスを提供します。XML-PI を使用する場合は、既知のキーワードを使用して IOS `show` コマンドの出力を Tcl スクリプトから解析できます。「スクリーンスクレイピング」出力に対する正規表現サポートを使用する必要はありません。

XML-PI コマンド拡張を使用する利点は、CLI `show` コマンドを使用して生成される特定の出力情報の抽出を容易にすることです。ほとんどの `show` コマンドは出力内の多くのフィールドを返しますが、現在のところ、行の中央に表示される可能性がある特定の情報を抽出するには正規表現を使用する必要があります。XML-PI サポートは一連の Tcl ライブラリ関数を提供し、次の形式の IOS CLI 形式の拡張からの出力の解析を容易にします。

`show`

```
<  
show-command  
> | format  
 {  
spec-file  
}
```

ここで、`spec-file` は現在サポートされている各 `show` コマンドのすべての SPEC ファイルエントリ (SFE) を連結したものです。XML-PI プロジェクトの一環として、デフォルトの `spec-file` が IOS リリース 12.4(22)T イメージに組み込まれます。デフォルトの `spec-file` には、一連の少数のコマンドが組み込まれ、それらのコマンドの SFE には考えられるタグのサブセットが組み込まれます。`format` コマンドで `spec-file` が提供されない場合、デフォルトの `spec-file` が使用されます。

XML-PI に関するより一般的な詳細については、「XML-PI」の章を参照してください。

翻訳について

このドキュメントは、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。