Embedded Event Manager ポリシーの設定および管理

Cisco IOS XR ソフトウェアの Embedded Event Manager（EEM）は、Cisco IOS XR ソフトウェア プロセッサのフェールオーバー サービスの任意の一部で検出されたイベントの中央クリア ハウスとして機能します。 EEM は、Cisco IOS XR ソフトウェア システム内の障害イベント、ディザスタ リカバリ、およびプロセス信頼性統計情報を担当します。 EEM イベントは、次のような重要なイベントが発生したことを示す通知です。

- 許容値を逸脱した運用統計情報またはパフォーマンス統計情報（たとえば、空きメモリがクリティカルなしきい値未満に低下したなど）。
- 活性挿抜（OIR）。
- プロセスの終了。

EEM は、ソフトウェア エージェントおまたはイベント ディテクタに依存して、特定のシステム イベントが発生したときに通知します。 イベントを検出すると、EEM は修正アクションを開始できます。 このアクションは、*policies* という名前のルーチンに規定されています。 収集されたイベントに対してアクションを適用できるようにするには、ポリシーを登録しておく必要があります。 ポリシーを登録しないと、アクションは発生しません。 登録されたポリシーにより、検出される特定のイベント、およびそのイベントが検出された場合に実行される修正アクションが EEM に通知されます。 このようなイベントが検出されると、EEM により対応するポリシーがイネーブル化されます。 登録されたポリシーは、いつでもディセーブルにできます。

EEM は、システムの各プロセスによって達成された信頼性の評価をモニタリングし、システムが全体的な信頼性または可用性を脅かすコンポーネントを検出できるようにします。

このモジュールでは、Cisco ASR 9000 シリーズ ルータで EEM ポリシーを設定して管理し、Tool Command Language（Tcl）スクリプトを使用して EEM ポリシーの書き込みおよびカスタマイズを実行してCisco IOS XR ソフトウェアの障害とイベントを処理するために必要な新規および改訂されたタスクについて説明します。

（注） このモジュールでリストされているイベント管理コマンドの詳細については、このモジュールの関連資料，（66 ページ）を参照してください。

Embedded Event Manager ポリシーの設定および管理の機能履歴

リリース	変更箇所
リリース 4.0.0	この機能が導入されました。

Embedded Event Manager ポリシーの設定および管理の前提条件

適切なタスク ID を含むタスク グループに関連付けられているユーザ グループに属している必要があります。 このコマンド リファレンスには、各コマンドに必要なタスク ID が含まれます。ユーザ グループの割り当てが原因でコマンドを使用できないと考えられる場合、AAA 管理者に連絡してください。

Embedded Event Manager ポリシーの設定および管理について

EEM ポリシーを実装するには、次の概念を理解する必要があります。

イベント管理

Cisco IOS XR ソフトウェア システムの Embedded Event Management（EEM）には、基本的にシステムイベント管理が含まれます。イベントは、システム内で起こった重要なオカレンス（エラー

に限らず）です。 Cisco IOS XR ソフトウェア EEM は、これらのイベントを検出して、適切な応答を実行します。 また、EEM を使用して、障害を防止または包含、およびディザスタ リカバリを支援することができます。

システム管理者は、EEMを使用して、システムの現在の状態に基づいて適切なアクションを指定できます。 たとえば、システム管理者は EEM を使用して、ハードウェア デバイスの交換が必要になったときに、電子メールによる通知を要求することができます。

また、EEM はシステムの各プロセスの信頼性メトリックも維持します。

システム イベント検出

EEMは、システムをモニタしてイベントを検出するルーチンである「イベントディテクタ」と相互作用します。 EEM は、syslog に提供したイベント ディテクタに依存して、特定のシステム イベントが発生したことを検出します。 EEMは、syslogメッセージとのパターンマッチを使用します。 また、タイマー イベント ディテクタにも依存して、特定の日時が発生したとこを検出します。

ポリシーベースのイベント応答

イベントを検出すると、EEM は応答アクションを開始できます。 これらのアクションは、*policy handlers* という名前のルーチンに含まれています。 イベント検出用のデータが収集されている間は、そのイベントに応答するポリシーが登録されるまでアクションは実行されません。 登録時には、ポリシーから EEM に、ポリシーが特定のイベントを検索していることが通知されます。 イベントを検出すると、EEM はポリシーをイネーブルにします。

信頼性メトリック

EEMは、システムの各プロセスによって達成された信頼性の評価を監視します。テスト中にこれらのメトリックを使用して、どのコンポーネントが信頼性または可用性の目標に到達していないかを特定し、修正アクションを実行できるようにすることが可能です。

システム イベント処理

イベント通知を受信すると、EEM は次のアクションを実行します。

- 確立されたポリシー ハンドラが存在するか確認します。
  - ポリシー ハンドラが存在する場合、EEMはコールバック ルーチン（*EEM* ハンドラ）を開始するか、Tool Command Language（Tcl）スクリプト（*EEM* スクリプト）を実行してポリシーを実装します。 ポリシーには、組み込み EEM アクションが含まれる場合があります。
  - ポリシー ハンドラが存在しない場合、EEM は何も実行しません。
- イベント通知に加入しているプロセスに通知します。

（注） ポリシー アクションが含まれるスクリプトと、イベントを受信するように加入しているスクリプトの間には違いがあります。 ポリシー アクションが含まれるスクリプトは、ポリシーを実装するものと想定されます。 これらは、再帰を防止するルールによってバインドされています。 通知に加入しているスクリプトは、このようなルールにバインドされることはありません。

- システム内の各プロセスの信頼性メトリック データを記録します。
- アプリケーション プログラム インターフェイス（API）を介して、EEM により維持されているシステム情報へのアクセスを提供します。

Embedded Event Manager 管理ポリシー

イベントを検出すると、EEM は修正アクションを開始できます。 このアクションは、*policies* という名前のルーチンに規定されています。ポリシーは、Tcl API を使用してユーザにより書き込まれた Tcl スクリプト（EEM スクリプト）によって定義されます。 （Embedded Event Manager スクリプトとスクリプティング インターフェイス（Tcl）， （4 ページ）を参照）。収集されたイベントに対してアクションを適用できるようにするには、ポリシーを登録しておく必要があります。ポリシーを登録しないと、アクションは発生しません。 登録されたポリシーにより、検出する特定のイベント、およびそのイベントが検出された場合に実行される修正アクションが EEM に通知されます。 このようなイベントが検出されると、EEM により対応するポリシーが実行されます。登録されたポリシーは、いつでもディセーブルにできます。

Embedded Event Manager スクリプトとスクリプティング インターフェイス（Tcl）

EEM スクリプトは、EEM イベントがパブリッシュされるときに、ポリシーを実装するために使用されます。 EEM スクリプトおよびポリシーは、**event manager policy** コンフィギュレーション コマンドを使用して EEM で識別されます。 EEM スクリプトは、**no event manager policy** コマンドが入力されない限り、EEM によるスケジューリングが可能なままになります。

EEM は、次の 2 つのタイプの EEM スクリプトを使用します。

- レギュラー EEM スクリプトは、**eem script** CLI コマンドを使用して EEM で識別されます。レギュラー EEM スクリプトはスタンドアロン スクリプトであり、このスクリプトが処理するイベントの定義を包含します。
- *EEM* コールバック スクリプトは、プロセスまたは EEM スクリプトがイベントを処理するように登録されたときに、EEM で識別されます。 EEM コールバック スクリプトは、基本的に名前付き関数で、C 言語 API を使用して EEM で識別されます。

次に、スクリプトでの CLI の使用例を示します。

```
sjc-cde-010:/tftpboot/cnwei/fm> cat test_cli_eem.tcl
::cisco::eem::event_register_syslog occurs 1 pattern $_syslog_pattern maxrun 90

namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

set errorInfo ""

# 1. query the information of latest triggered fm event
array set arr_einfo [event_reqinfo]

if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}

set msg $arr_einfo(msg)
set config_cmds ""

# 2. execute the user-defined config commands
if [catch {cli_open} result] {
    error $result $errorInfo
} else {
    array set cli1 $result
}

if [catch {cli_exec $cli1(fd) "config"} result] {
    error $result $errorInfo
}

if {[info exists _config_cmd1]} {
    if [catch {cli_exec $cli1(fd) $_config_cmd1} result] {
        error $result $errorInfo
    }

    append config_cmds $_config_cmd1
}

if {[info exists _config_cmd2]} {
    if [catch {cli_exec $cli1(fd) $_config_cmd2} result] {
        error $result $errorInfo
    }
    append config_cmds "\n"
    append config_cmds $_config_cmd2
}

if [catch {cli_exec $cli1(fd) "end"} result] {
    error $result $errorInfo
}
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
    error $result $errorInfo
}

action_syslog priority info msg "Ran config command $_config_cmd1 $_config_cmd2
```

スクリプト言語

スクリプト言語は、Cisco IOS XR ソフトウェアに実装されている Tool Command Language（Tcl）です。 すべての Embedded Event Manager スクリプトは、Tcl で記述されます。 この Tcl のフル実装はシスコによって拡張され、 **eem** コマンドが追加されて Tcl スクリプトと EEM の間にインターフェイスが提供されました。

Tcl は文字列ベースのコマンド言語で、実行時に解釈されます。 Tcl がサポートされるバージョンは、Tcl バージョン 8.3.4 に、スクリプト サポートが追加されたものです。 スクリプトは、ネットワーキング デバイスではなく、別のデバイスで、ASCII エディタを使用して定義されます。 続いてスクリプトはネットワーキング デバイスにコピーされ EEM に登録されます。 Tcl スクリプトは EEM でサポートされます。 強制適用される規則としての Embedded Event Manager ポリシーは、経過時間 20 秒未満で解釈および実行される必要がある、存続時間の短い実行時ルーチンです。 20 秒よりも長い経過時間が必要な場合、event_register 文で maxrun パラメータを使用して、必要な値を指定する必要があります。

EEM ポリシーでは、すべての Tcl 言語機能が使用されます。 ただし、シスコでは、EEM ポリシーの記述に活用できる Tcl コマンド拡張の形式で、Tcl 言語の機能を拡張しています。 Tcl コマンド拡張のキーワードの主要なカテゴリでは、検出されたイベント、後続のアクション、ユーティリティ情報、カウンタの値、システム情報が特定されます。

EEM では、Tcl を使用して独自のポリシーを記述、実装できます。 EEM スクリプトの記述には、次の作業が含まれます。

- ポリシーの実行時に決定に使用される基準を確立する、イベント Tcl コマンド拡張の選択。
- イベントの検出に関連付けられているイベント ディテクタ オプションの定義。
- 検出されたイベントのリカバリまたは検出されたイベントに対する応答を実装するアクションを選択すること。

レギュラー Embedded Event Manager スクリプト

レギュラー EEM スクリプトは、EEM イベントがパブリッシュされるときに、ポリシーを実装するために使用されます。 EEM スクリプトは、**event manager policy** コンフィギュレーション コマンドを使用して EEM で識別されます。 EEM スクリプトは、**no event manager policy** コマンドが入力されない限り、EEM によるスケジューリングが可能なままになります。

EEM スクリプト内の最初コード実行可能行は、**eem event register** キーワードである必要があります。 このキーワードは、スクリプトのスケジュール対象となる EEM イベントを特定します。 このキーワードは、指定された EEM イベントを処理するように登録するために、**event manager policy** コンフィギュレーション コマンドにより使用されます。

EEM スクリプトでは、Embedded Event Manager ポリシー Tcl コマンド拡張カテゴリ，（7 ページ）にリストされている任意の EEM スクリプト サービスが使用される場合があります。

EEM スクリプトが存在する場合、この EEM イベントのデフォルト アクション（存在する場合）を実行するか、その他のアクションを実行しないかを EEM に通知するために使用される戻りコードの設定を担当します。 特定のイベントに対して複数のイベント ハンドラがスケジューリングされている場合、前のハンドラからの戻りコードが次のハンドラに渡されます。これにより、値をそのままにするか、更新できます。

（注） EEM スクリプトは、スケジュールの対象となるイベント以外のイベントを処理するように登録することはできません。

Embedded Event Manager コールバック スクリプト

EEM コールバック スクリプトは、eem_handler_spec でこのスクリプトの名前を指定している、以前に登録された EEM イベントに対して発生している EEM イベントの結果として入力されます。

EEM コールバック スクリプトが存在する場合、この EEM イベントのデフォルト アクション（存在する場合）を実行するかどうかを EEM に通知するために使用される戻りコードの設定を担当します。特定のイベントに対して複数のイベント ハンドラがスケジューリングされている場合、前のハンドラからの戻りコードが次のハンドラに渡されます。これにより、値をそのままにするか、更新できます。

（注） EEM コールバック スクリプトは、表 1：Embedded Event Manager Tcl コマンド拡張カテゴリ，（7 ページ）にリストされている EEM スクリプトを自由に使用できます。ただし、**eem event register** キーワードは EEM コールバック スクリプトでは許可されていないため、使用できません。

Embedded Event Manager ポリシー Tcl コマンド拡張カテゴリ

この表には、EEM ポリシー Tcl コマンド拡張のさまざまなカテゴリの一覧を示します。

（注） すべての EEM ポリシーで使用するこれらの各カテゴリで使用可能な Tcl コマンドは、この資料の以降の項で説明します。

表 *1*：*Embedded Event Manager Tcl* コマンド拡張カテゴリ

カテゴリ	定義
EEM イベントの Tcl コマンド拡張（イベント情報、イベント登録、イベント パブリッシュの 3 タイプ）	これらの Tcl コマンド拡張は、**event_register**_xxx ファミリのイベント固有コマンドによって表されます。このカテゴリには、別のイベント情報 Tcl コマンド拡張の **event_reqinfo** もあります。これは、イベントについての情報を EEM に問い合わせるためにポリシーで使用されるコマンドです。アプリケーション固有のイベントをパブリッシュする、EEM イベント Tcl コマンド拡張 **event_publish** もあります。

カテゴリ	定義
EEM アクションの Tcl コマンド拡張	これらの Tcl コマンド拡張（たとえば、**action_syslog** など）は、イベントまたは障害への応答か、または、イベントまたは障害からの回復のために、ポリシーによって使用されます。 これらの拡張に加え、開発者は、Tcl 言語を使用して、必要なアクションを実装できます。
EEM ユーティリティの Tcl コマンド拡張	これらの Tcl コマンド拡張は、アプリケーション情報、カウンタ、またはタイマーの取得、保存、設定、または変更に使用されます。
EEM システム情報の Tcl コマンド拡張	これらの Tcl コマンド拡張は、**sys_reqinfo**_xxx ファミリのシステム固有情報コマンドによって表されます。 これらのコマンドは、システム情報を収集する目的で、ポリシーによって使用されます。
EEM コンテキストの Tcl コマンド拡張	これらの Tcl コマンド拡張は、Tcl コンテキスト（可視変数およびその値）の保存および取得に使用されます。

Embedded Event Manager 用のシスコ ファイル命名規則

すべての EEM ポリシー名、ポリシー サポート ファイル（たとえば、電子メール テンプレート ファイル）、およびライブラリ ファイル名は、シスコのファイル命名規則に従う必要があります。 これに関連し、EEM ポリシーのファイル名は次の仕様に従います。

- オプションのプレフィックス Mandatory. がある場合、これは、システム ポリシーがまだ登録されていない場合に、自動的に登録される必要があるシステム ポリシーであることを示します（たとえば Mandatory.sl_text.tcl）。
- 指定された 1 つめのイベントの 2 文字の省略形が含まれるファイル名の本体部（表 2：2 文字の省略形の指定，（9 ページ） を参照）、下線部、および、ポリシーをさらに示す説明フィールド部。
- ファイル名拡張子部は .tcl と定義されます。

EEM の電子メール テンプレート ファイルは、email_template のファイル名のプレフィックスと、後続の電子メール テンプレートの使用状況を示す省略形で構成されます。

EEM ライブラリ ファイル名は、ライブラリの使用状況を示す説明フィールドを含むファイル名の本体部と、後続の _lib、および .tcl というファイル名拡張子で構成されます。

表 *2* : *2* 文字の省略形の指定

2 文字の省略形	仕様
ap	event_register_appl
ct	event_register_counter
st	event_register_stat
no	event_register_none
oi	event_register_oir
pr	event_register_process
rf	event_register_rf
sl	event_register_syslog
tm	event_register_timer
ts	event_register_timer_subscriber
wd	event_register_wdsysmon

Embedded Event Manager の組み込みアクション

EEM の組み込みアクションは、EEM ハンドラが動作するときにハンドラから要求できます。

次の表に、各 EEM ハンドラ要求またはアクションを示します。

表 *3* : *Embedded Event Manager* の組み込みアクション

Embedded Event Manager の組み込みアクション	説明
syslog へのメッセージの記録	メッセージを syslog に送ります。 このアクションに対する引数は、優先度と記録するメッセージです。
CLI コマンドの実行	指定されたチャネルハンドラにコマンドを書き込み、**cli_exec** コマンド拡張を使用してコマンドを実行します。
syslog メッセージの生成	**action_syslog** Tcl コマンド拡張を使用して、メッセージをログに記録します。

Embedded Event Manager の組み込みアクション	説明
手動による EEM ポリシーの実行	**event manager run** コマンドが EXEC モードでポリシーを実行している間に、ポリシー内で EEM ポリシーを実行します。
アプリケーション固有のイベントのパブリッシュ	**event_publish appl** Tcl コマンド拡張を使用して、アプリケーション固有のイベントをパブリッシュします。
Cisco IOS ソフトウェアのリロード	EEM **action_reload** コマンドを使用して、ルータをリロードします。
システム情報の要求	システム情報を収集するための、ポリシーによる sys_reqinfo_xxx ファミリのシステム固有情報コマンドを表します。
ショート メールの送信	シンプル メール転送プロトコル（SMTP）を使用して、電子メールを送信します。
カウンタの設定または変更	カウンタの値を変更します。

EEM ハンドラは、CLI コマンドを実行できる必要があります。 Tcl スクリプトの中からの CLI コマンドの実行を許可するために、Tcl シェルでコマンドを使用できます。

アプリケーション固有の組み込みイベント管理

どの Cisco IOS XR ソフトウェア アプリケーションも、アプリケーション定義のイベントを定義してパブリッシュできます。 アプリケーション定義のイベントは、コンポーネント名とイベント名の両方を含む名前で識別され、アプリケーション開発者が独自のイベント ID を割り当てることができます。 アプリケーションで定義されたイベントは、サブスクライバがいない場合でも、Cisco IOS XR ソフトウェア コンポーネントによって発行できます。 この場合、イベントは EEM によって解除されるため、サブスクライバはアプリケーション定義のイベントを必要に応じて受信できます。

システム イベントを受信するためにサブスクライブする EEM スクリプトは、次の順序で処理されます。

1. CLI コンフィギュレーション コマンド **event manager policy** *scriptfilename* **username** *username* が入力されます。
2. EEM は EEM スクリプトをスキャンして **eem event event_type** キーワードを探し、指定したイベントに対してスケジュールされるように EEM スクリプトをサブスクライブします。
3. イベント ディテクタがイベントを検出し、EEM に通知します。

4 EEM はイベント処理をスケジュールし、EEM スクリプトが実行されます。

5 EEM スクリプト ルーチンが戻ります。

イベント検出とリカバリ

イベントは、イベント ディテクタと呼ばれるルーチンによって検出されます。 イベント ディテクタは、他の Cisco IOS XR ソフトウェア コンポーネントと EEM の間のインターフェイスを提供する個別のプログラムです。イベントディテクタは、必要に応じてイベントをパブリッシュするために使用可能な情報を処理します。

以下のイベント ディテクタがサポートされています。

EEMイベントは、システム内で何か重要なことが起きたことを示す通知として定義されます。イベントには次の 2 つのカテゴリがあります。

- システム EEM イベント
- アプリケーション定義イベント

システム EEM イベントは EEM に組み込まれており、イベントを発生する障害ディテクタに基づいてグループ化されます。 API の中で定義されたシンボリックな ID で識別されます。

一部の EEM システム イベントは、アプリケーションがモニタリングを要求したかどうかにかかわらず EEM によってモニタされます。 そのようなイベントを、組み込み EEM イベントと呼びます。 他の EEM イベントは、アプリケーションが EEM イベント モニタリングを要求した場合のみモニタされます。 EEM イベント モニタリングは、EEM アプリケーション API または EEM スクリプティング インターフェイスを通じて要求されます。

一部のイベントディテクタは、同じセキュア ドメイン ルータ（SDR）または管理プレーンの中の他のハードウェア カードに分散させて、それらのカード上で動作する分散コンポーネントをサポートできます。

EEM イベントの検出および回復の一般的なフロー

EEM は、イベント ディテクタと呼ばれるソフトウェア エージェントを使用してシステム内の異なるコンポーネントのモニタリングをサポートする、柔軟でポリシードリブンのフレームワークです。 EEM サーバ、コア イベント パブリッシャ（イベント ディテクタ）、およびイベント サブスクライバ（ポリシー）の間にな関係があります。イベントパブリッシャはイベントを選別し、イベント サブスクライバによって提供されたイベント仕様と一致するときに、それらをパブリッシュします。 イベント ディテクタは、注目するイベントが発生したときに EEM サーバに通知します。

イベントまたは障害が検出されると、Embedded Event Manager によって、たとえば OIR イベント パブリッシャなどのイベント パブリッシャから、検出された障害またはイベントの登録が発生しているかどうかが判断されます。 EEM によって、イベント登録情報が、イベント データそのものと、照会されます。 ポリシーによって、検出されたイベントが Tcl コマンド拡張 event_register_xxx

で登録されます。 イベント情報 Tcl コマンド拡張 event_reqinfo は、検出されたイベントに関する情報について Embedded Event Manager に問い合わせるために、ポリシーで使用されます。

System Manager イベント ディテクタ

System Manager イベント ディテクタには、次の 4 つの役割があります。

- プロセス信頼性メトリック データを記録します。
- 未処理の EEM イベント モニタリング要求があるプロセスをスクリーニングします。
- スクリーニング条件に一致するプロセスのためのイベントをパブリッシュします。
- スクリーニング条件に一致しないイベントについて、デフォルトのアクションを実行するように System Manager に依頼します。

System Manager イベント ディテクタは、System Manager と通信して、プロセスの起動通知と終了通知を受信します。 この通信は、System Manager が使用可能なプライベート API を通じて行われます。 オーバーヘッドを最小化するため、API の一部は System Manager プロセス空間の中にあります。 プロセスが終了するとき、System Manager は、イベント ディテクタ API を呼び出す前に、ヘルパー プロセスを起動します（process.startup ファイルで指定されている場合）。

プロセスはコンポーネント ID、System Manager によって割り当てられたジョブ ID、またはロード モジュールのパス名にプロセス インスタンス ID を加えたもので識別されます。 *、?、または [...] を使用した POSIX ワイルドカード ファイル名パターンがロード モジュールのパス名でサポートされています。 プロセス インスタンス ID は、プロセスを同じパス名の他のプロセスと区別するために割り当てられる整数です。 プロセスの最初のインスタンスにはインスタンス ID 値 1 が割り当てられ、次に 2 というように割り当てられます。

System Manager イベント ディテクタは、次の表に示す EEM イベントの EEM イベント モニタリング要求を処理します。

表 *4*：*System Manager* イベント ディテクタ イベント モニタリング要求

Embedded Event Manager イベント	説明
プロセス正常終了 EEM イベント：組み込み	スクリーニング条件に一致するプロセスが終了するときに発生します。
プロセス異常終了 EEM イベント：組み込み	スクリーニング条件に一致するプロセスが異常終了するときに発生します。
プロセス起動 EEM イベント：組み込み	スクリーニング条件に一致するプロセスが起動するときに発生します。

System Manager イベント ディテクタ プロセス異常終了イベントが発生した場合、デフォルトのアクションにより、System Manager の組み込み規則に従ってプロセスが再起動されます。

EEM と System Manager の間の関係は、プロセスの起動通知と終了通知を受信する目的で EEM により System Manager に提供されているプライベート API を通じて厳格に行われます。 System Manager が API を呼び出すとき、信頼性メトリック データが収集され、EEM イベント一致のためにスクリーニングが実行されます。 一致が見つかった場合、System Manager イベント ディテクタにメッセージが送信されます。プロセス異常終了の場合、EEM がプロセスの再起動を処理することを通知して戻ります。 一致しない場合、System Manager がデフォルトのアクションを適用すべきことを通知して戻ります。

タイマー サービス イベント ディテクタ

タイマー サービス イベント ディテクタは、時間に関連する EEM イベントを実装します。 これらのイベントは、複数のプロセスが同じ EEM イベントの通知を待つことができるように、ユーザ定義 ID を通じて識別されます。

タイマー サービス イベント ディテクタは、日付/時刻経過 EEM イベントの EEM イベント モニタリング要求を処理します。 このイベントは、現在の日付または時刻が、アプリケーションによって要求された指定の日付または時刻を過ぎた場合に発生します。

syslog イベント ディテクタ

syslog イベント ディテクタは、syslog EEM イベントのための syslog メッセージ スクリーニングを実現します。 このルーチンは、プライベート API を通じて syslog デーモンと通信します。 オーバーヘッドを最小化するため、API の一部は syslog デーモン プロセスの中にあります。

メッセージ重大度コードまたはメッセージテキスト フィールドに対するスクリーニング機能を利用できます。 メッセージ テキスト フィールドでは POSIX 正規表現パターンがサポートされています。

syglog イベント ディテクタは、次の表に示すイベントの EEM イベント モニタリング要求を処理します。

表 *5*：*syslog* イベント ディテクタイベント モニタリング要求

Embedded Event Manager イベント	説明
syslog メッセージ EEM イベント	ログに記録されたばかりのメッセージに対して発生します。 syslog メッセージの重大度コードまたは syslog メッセージ テキスト パターンのいずれかが一致した場合に発生します。いずれも、アプリケーションが syslog メッセージ EEM イベントを要求するときに指定できます。
プロセス イベント マネージャ EEM イベント：組み込み	指定したプロセスのイベント処理カウントが指定した最大値以上になるか、指定した最小値以下になった場合に発生します。

None イベント ディテクタ

None イベント ディテクタは、Cisco IOS XR ソフトウェア**event manager run** CLI コマンドが EEM ポリシーを実行したときにイベントをパブリッシュします。 EEMは、ポリシーそのものに含まれるイベント仕様に基づいてポリシーをスケジューリングし、実行します。 EEMポリシーは識別される必要があり、手動での実行が許可されるように、**event manager run** コマンドが実行される前に登録される必要があります。

イベントマネージャのNoneディテクタを使用すると、CLIを使用してTclスクリプトを実行できます。 スクリプトは、実行前に登録します。 Cisco IOS XR ソフトウェア バージョンは、Cisco IOS EEMと同様の構文を備えているため（詳細は該当するEEMのマニュアルを参照してください）、Cisco IOS EEM を使用して作成したスクリプトは、最小限の変更により Cisco IOS XR ソフトウェアで実行できます。

Watchdog System Monitor イベント ディテクタ

Cisco IOS XR ソフトウェア用の Watchdog System Monitor（IOSXRWDSysMon）イベント ディテクタ

Cisco IOS XR ソフトウェア Watchdog System Monitor イベント ディテクタは、次のいずれかが発生したときにイベントをパブリッシュします。

- Cisco IOS XR ソフトウェア プロセスでの CPU の利用率がしきい値を超えたとき。
- Cisco IOS XR ソフトウェア プロセスでのメモリの利用率がしきい値を超えたとき。

（注） Cisco IOS XR ソフトウェア プロセスは、Cisco IOS XR ソフトウェア モジュール方式プロセスと区別するために使用されます。

同時に 2 つのイベントがモニタリングされることがあります。指定されたしきい値を超えるために 1 つのイベントを必要とするか、両方のイベントを必要とするかを、イベント パブリッシング基準で指定できます。

Cisco IOS XR ソフトウェア Watchdog System Monitor イベント ディテクタは、以下の表に示すイベントを処理します。

表 *6*：*Watchdog System Monitor* イベント ディテクタ要求

Embedded Event Manager イベント	説明
プロセス パーセント CPU EEM イベント：組み込み	指定したプロセスのCPU時間が、使用可能CPU時間の指定した最大パーセンテージ以上になるか、使用可能 CPU 時間の指定した最小パーセンテージ以下になった場合に発生します。

Embedded Event Manager イベント	説明
合計パーセント CPU EEM イベント：組み込み	指定したプロセッサ コンプレックスの CPU 時間が、使用可能 CPU 時間の指定した最大パーセンテージ以上になるか、使用可能 CPU 時間の指定した最小パーセンテージ以下になった場合に発生します。
プロセス パーセント メモリ EEM イベント：組み込み	指定したプロセスで使用されているメモリが、指定した値だけ増加または減少した場合に発生します。
合計パーセント使用可能メモリ EEMイベント：組み込み	指定したプロセッサ コンプレックスの使用可能メモリが、指定した値だけ増加または減少した場合に発生します。
合計パーセント使用メモリ EEM イベント：組み込み	指定したプロセッサ コンプレックスの使用メモリが、指定した値だけ増加または減少した場合に発生します。

Cisco IOS XR ソフトウェア モジュール方式のための Watchdog System Monitor（WDSysMon）イベント ディテクタ

Cisco IOS XR ソフトウェア ソフトウェア モジュール方式 Watchdog System Monitor イベント ディテクタは、Cisco IOS XR ソフトウェア モジュール方式プロセスの無限ループ、デッドロック、メモリ リークを検出します。

分散イベント ディテクタ

EEM イベント ディテクタと通信し、非常に独立した実装を持つ、分散したハードウェア カード上で動作する Cisco IOS XR ソフトウェア コンポーネントには、分散 EEM イベント ディテクタが必要です。 分散イベント ディテクタでは、EEM 通信チャネルへのローカル ハードウェア カードをアクティブにしなくても、ローカル プロセスの EEM イベントをスケジューリングできます。

次のイベント ディテクタが Cisco IOS XR ソフトウェア ラインカードで動作します。

- System Manager 障害ディテクタ
- Wdsysmon 障害ディテクタ
- Counter イベント ディテクタ
- OIR イベント ディテクタ
- Statistic イベント ディテクタ

Embedded Event Manager イベントのスケジューリングおよび通知

EEM ハンドラがスケジュールされている場合、EEM ハンドラは、イベント要求を作成するプロセスのコンテキスト（EEM スクリプトの場合は、Tcl シェル プロセス コンテキスト）で動作します。 EEM ハンドラを実行するプロセスに対して発生するイベントの場合、イベント スケジューリングは、ハンドラが終了するまでブロックされます。 代わりに、定義されたデフォルトのアクション（存在する場合）が実行されます。

EEM サーバは、要求された場合に、クライアント プロセスの再起動にまたがって、イベント スケジューリングと通知項目が格納されたキューを保持します。

信頼性統計情報

プロセッサ コンプレックス全体の信頼性メトリック データが EEM によって保持されています。 データはチェックポイントに定期的に書き込まれます。

ハードウェア カードの信頼性メトリック データ

信頼性メトリック データは、プロセッサ コンプレックスの各ハードウェア カードについて保持されます。 データは、ディスク ID でインデックスが作成されたテーブルに記録されます。

ハードウェア カードで保持されているデータは次のとおりです。

- 最新の起動時刻
- 最新の正常終了時刻（制御されたスイッチオーバー）
- 最新の異常終了時刻（非同期スイッチオーバー）
- 最新の異常のタイプ
- 累積使用可能時間
- 累積使用不能時間
- ハードウェア カード起動回数
- ハードウェア カード正常シャットダウン回数
- ハードウェア カード異常シャットダウン回数

プロセス信頼性メトリック データ

信頼性メトリック データは、System Manager によって処理される各プロセスについて保持されます。 このデータには、プライマリまたはバックアップ ハードウェア カードで動作するスタンバイ プロセスが含まれています。 データは、ハードウェア カード ディスク ID、プロセス パス名、複数のインスタンスがあるプロセスの場合はプロセス インスタンスを組み合わせたものでインデックスが作成されたテーブルに記録されます。

プロセスの終了には次の場合が含まれます。

- 正常終了：プロセスは終了値 0 で終了します。
- プロセスによる異常終了：プロセスは 0 でない終了値で終了します。
- QNX による異常終了：Neutrino オペレーティング システムがプロセスを異常終了させます。
- プロセス終了 API によるプロセス異常終了：プロセス終了 API によりプロセスが終了します。

プロセスが保持するデータは次のとおりです。

- 最新のプロセス起動時刻
- 最新のプロセス正常終了時刻
- 最新のプロセス異常終了時刻
- 最新のプロセス異常終了のタイプ
- 以前の 10 回のプロセス終了時刻とタイプ
- 累積プロセス使用可能時間
- 累積プロセス使用不能時間
- 累積プロセス実行時間（プロセスが実際に CPU で動作した時間）
- 起動回数
- 正常終了回数
- 異常終了回数
- 過去 60 分間の異常障害回数
- 過去 24 時間の異常障害回数
- 過去 30 日間の異常障害回数

Embedded Event Manager ポリシーの設定および管理方法

ここでは、次の手順について説明します。

環境変数の設定

EEM 環境変数は、ポリシーの実行前にポリシーに対して外部定義された Tcl グローバル変数です。EEM ポリシー エンジンは、障害およびその他のイベントが発生したときに通知を受け取ります。EEM ポリシーは、システムの現在の状態に基づいて回復を実行し、該当するイベントのポリシーに指定されたアクションを実行します。 回復アクションはポリシーが実行されたときにトリガーされます。

環境変数

通例として、シスコで定義されているすべての環境変数の名前は、他の変数と区別するためにアンダースコア文字で始まります（_show_cmd など）。

event manager environment コマンドの *var-value* 引数ではスペースを使用できます。 このコマンドは、*var-name* 引数の後から行の最後までのすべての文字列を *var-value* 引数の一部として解釈します。

event manager environment コマンドを使用して設定された後に、すべての EEM 環境変数の名前および値を表示するには、**show event manager environment** コマンドを使用します。

手順の概要

1. **show event manager environment**
2. **configure**
3. **event manager environment** *var-name var-value*
4. リセットするすべての環境変数について手順 3 を繰り返します。
5. 次のいずれかのコマンドを使用してください。
   - **end**
   - **commit**
6. **show event manager environment**

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**show event manager environment** 例： `RP/0/RSP0/CPU0:router# show event manager environment`	すべての EEM 環境変数の名前と値を表示します。
ステップ 2	**configure** 例： `RP/0/RSP0/CPU0:router# configure`	グローバル コンフィギュレーション モードを開始します。
ステップ 3	**event manager environment** *var-name var-value* 例： `RP/0/RSP0/CPU0:router(config)# event manager environment _cron_entry 0-59/2 0-23/1 * * 0-7`	環境変数を新しい値にリセットします。 • *var-name* 引数は、EEM 環境設定変数に割り当てる名前です。 • *var-value* 引数は、環境変数 *var-name* に格納する、スペースを含む文字列です。

	コマンドまたはアクション	目的
		• 通例として、シスコで定義されているすべての環境変数の名前は、他の変数と区別するためにアンダースコア文字で始まります（_show_cmd など）。 • *var-value* 引数では、スペースを使用できます。 このコマンドは、*var-name* 引数の後から行の最後までのすべての文字列を *var-value* 引数の一部として解釈します。
ステップ 4	リセットするすべての環境変数について手順 3 を繰り返します。	—

ステップ 5	次のいずれかのコマンドを使用してください。
	• **end**
	• **commit**
	例：
	`RP/0/RSP0/CPU0:router(config)# end`
	または
	`RP/0/RSP0/CPU0:router(config)# commit`
		設定変更を保存します。
		• **end** コマンドを実行すると、変更をコミットするように要求されます。
		`Uncommitted changes found, commit them before exiting(yes/no/cancel)? [cancel]:`
		◦ yes と入力すると、実行コンフィギュレーション ファイルに変更が保存され、コンフィギュレーション セッションが終了して、ルータが EXEC モードに戻ります。
		◦ no と入力すると、コンフィギュレーション セッションが終了して、ルータが EXEC モードに戻ります。変更はコミットされません。
		◦ cancel と入力すると、現在のコンフィギュレーションセッションが継続します。コンフィギュレーション セッションは終了せず、設定変更もコミットされません。
		• 実行コンフィギュレーション ファイルに設定変更を保存し、コンフィギュレーション セッションを継続するには、**commit** コマンドを使用します。

ステップ 6	**show event manager environment** 例： `RP/0/RSP0/CPU0:router# show event manager environment`	すべての EEM 環境変数のリセットされた名前と値を表示します。手順 3 で設定した環境変数名と値を確認できます。

次の作業

EEM 環境変数を設定した後、Embedded Event Manager ポリシーの登録, （20 ページ）に従って、登録できるポリシーを確認し、ポリシーを登録します。

Embedded Event Manager ポリシーの登録

イベントがトリガーされたときにポリシーを実行するため、EEM ポリシーを登録します。

Embedded Event Manager ポリシー

EEM ポリシーを登録するには、グローバル コンフィギュレーション モードで **event manager policy** コマンドを使用します。 EEM スクリプトは、このコマンドの **no** 形式が入力されない限り、EEM によるスケジューリングが可能です。 ポリシーを登録する前に、**show event manager policy available** コマンドを使用して、登録できる EEM ポリシーを表示します。

EEM は、ポリシー自体に含まれているイベントの指定内容に基づいて、ポリシーをスケジューリングおよび実行します。 **event manager policy** コマンドが呼び出されると、EEM はポリシーを確認し、指定されたイベントが発生した場合に実行されるように登録します。

Username
EEM ポリシーを登録するには、スクリプトを実行するために使用するユーザ名を指定する必要があります。 この名前は、現在ログインしているユーザと異なる名前でもかまいません。ただし、登録するユーザは、スクリプトを実行するユーザ名のスーパーセットである権限を所有している必要があります。所有していない場合、スクリプトは登録されず、コマンドは拒否されます。また、スクリプトを実行するユーザ名は、登録される EEM ポリシーが実行するコマンドへのアクセス権を所有している必要があります。

（注） EEM ポリシーを登録する前に、AAA 認可（**aaa authorization eventmanager** コマンドなど）を設定する必要があります。 AAA 認可の設定の詳細については、『*Configuring AAA Services on Cisco IOS XR* ソフトウェア』の「*Configuring AAA Services*」モジュールを参照してください。

持続時間
ユーザ名に対するオプションの **persist-time** キーワードも定義できます。 **persist-time** キーワードは、ユーザ名認証が有効な時間（秒数）を定義します。 スクリプトの初回登録時は、スクリプトに対して設定された username が認証されます。 スクリプトの登録後は、スクリプトの実行ごとにユーザ名が再度認証されます。 AAA サーバがダウンしている場合は、ユーザ名の認証をメモリから読み取れます。 このユーザ名の認証をメモリに保持する秒数は、**persist-time** キーワードによって決定します。

- AAA サーバがダウンしていて **persist-time** キーワードが期限切れになっていない場合、ユーザ名はメモリから認証され、スクリプトが実行されます。
- AAA サーバがダウンしていて **persist-time** キーワードが期限切れの場合、ユーザ認証が失敗して、スクリプトは実行されません。

persist-time キーワードには、次の値を使用できます。

- デフォルトの **persist-time** は、3600 秒（1 時間）です。 **persist-time** を 1 時間に設定するには、**persist-time** キーワードを指定せずに **event manager policy** コマンドを入力します。
- ユーザ名の認証がキャッシュされないようにするには、0を指定します。 AAAサーバがダウンしている場合、ユーザ名は認証されず、スクリプトは実行されません。
- ユーザ名が無効としてマーキングされないようにするには、**infinite** を指定します。 キャッシュに保持されたユーザ名の認証は、期限切れになりません。 AAA サーバがダウンしている場合、ユーザ名はキャッシュから認証されます。

system または user キーワード

system または **user** キーワードを指定せずに **event manager policy** コマンドを入力すると、指定されたポリシー ファイルが、まずシステム ポリシー ディレクトリで検索されます。 システム ポリシーディレクトリ内でファイルが見つかった場合、そのポリシーがシステムポリシーとして登録されます。 指定されたポリシー ファイルがシステム ポリシー ディレクトリ内で見つからなかった場合、ユーザ ポリシー ディレクトリが検索されます。 指定されたファイルがユーザ ポリシーディレクトリ内で見つかった場合、そのポリシー ファイルがユーザ ポリシーとして登録されます。 同じ名前を持つポリシー ファイルがシステム ポリシー ディレクトリとユーザ ポリシー ディレクトリの両方で見つかった場合、システム ポリシー ディレクトリ内のポリシー ファイルが優先され、システム ポリシーとして登録されます。

ポリシーを登録した後でその登録内容を確認するには、**show event manager policy registered** コマンドを使用します。 出力では、登録済みポリシーの情報が 2 つの部分にわかれて表示されます。各ポリシーの説明の最初の行には、ポリシーに割り当てられているインデックス番号、ポリシーのタイプ（システムまたはユーザ）、登録済みイベントのタイプ、ポリシーの登録日時、およびポリシーファイルの名前が表示されます。各ポリシーの説明の残りの行には、登録済みイベントとイベントの処理方法に関する情報が表示されます。この情報は、ポリシー ファイルを構成する Tcl コマンドの引数から直接取得されます。

手順の概要

1. **show event manager policy available** [**system** | **user**]
2. **configure**
3. **event manager policy** *policy-name* **username** *username* [**persist-time** { *seconds* | **infinite** }] | **type** { **system** | **user** }
4. 登録するすべての EEM ポリシーについて手順 3 を繰り返します。
5. 次のいずれかのコマンドを使用してください。
   - **end**
   - **commit**
6. **show event manager policy registered**

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**show event manager policy available** [**system** | **user**] 例： `RP/0/RSP0/CPU0:router# show event manager policy available`	登録可能なすべての EEM ポリシーを表示します。 • オプションの **system** キーワードを入力すると、使用可能なすべてのシステム ポリシーが表示されます。 • オプションの **user** キーワードを入力すると、使用可能なすべてのユーザ ポリシーが表示されます。
ステップ 2	**configure** 例： `RP/0/RSP0/CPU0:router# configure`	グローバル コンフィギュレーション モードを開始します。
ステップ 3	**event manager policy** *policy-name* **username** *username* [**persist-time** { *seconds* | **infinite** }] | **type** { **system** | **user** } 例： `RP/0/RSP0/CPU0:router(config)# event manager policy cron.tcl username tom type user`	EEM ポリシーを EEM に登録します。 • EEM スクリプトは、このコマンドの **no** 形式が入力されない限り、EEM によるスケジューリングが可能です。 • 必要な **username** キーワードと引数を入力します。ここで、*username* はスクリプトを実行するユーザ名です。 • ユーザ名認証をメモリに保持する時間を指定するには、オプションの **persist-time** キーワードを入力します。 ◦ **persist-time** キーワードに *seconds*（秒数）を入力します。 ◦ 認証を永続的なものにするには、**infinite** キーワードを入力します（認証は期限切れになりません）。 • オプションの **type system** キーワードを入力すると、シスコによって定義されたシステム ポリシーが登録されます。 • オプションの **type user** キーワードを入力すると、ユーザ定義ポリシーが登録されます。 （注） EEM ポリシーを登録する前に、AAA 認可（aaa authorization eventmanage など）を設定する必要があります。 AAA 認可の設定の詳細については、 *Cisco ASR 9000 Series Aggregation Services Router System Security Configuration Guide*の「*Configuring AAA Services* 」モジュールを参照してください。
ステップ 4	登録するすべての EEM ポリシーについて手順 3 を繰り返します。	—

	コマンドまたはアクション	目的

ステップ 5	次のいずれかのコマンドを使用してください。
	• **end**
	• **commit**
	
	例：
	`RP/0/RSP0/CPU0:router(config)# end`
	または
	`RP/0/RSP0/CPU0:router(config)# commit`
		設定変更を保存します。
		• **end** コマンドを実行すると、変更をコミットするように要求されます。
		`Uncommitted changes found, commit them before exiting(yes/no/cancel)? [cancel]:`
		◦ yes と入力すると、実行コンフィギュレーション ファイルに変更が保存され、コンフィギュレーション セッションが終了して、ルータが EXEC モードに戻ります。
		◦ no と入力すると、コンフィギュレーションセッションが終了して、ルータが EXEC モードに戻ります。変更はコミットされません。
		◦ cancel と入力すると、現在のコンフィギュレーション セッションが継続します。コンフィギュレーション セッションは終了せず、設定変更もコミットされません。
		• 実行コンフィギュレーション ファイルに設定変更を保存し、コンフィギュレーションセッションを継続するには、**commit** コマンドを使用します。

ステップ 6	**show event manager policy registered** 例： `RP/0/RSP0/CPU0:router# show event manager policy registered`	登録済みのすべての EEM ポリシーを表示します。手順 3 の確認を行うことができます。

Tcl を使用した Embedded Event Manager ポリシーの記述方法

ここでは、Tool Command Language（Tcl）スクリプトを使用して、Cisco IOS XR ソフトウェアの障害とイベントを処理するための、カスタマイズした Embedded Event Manager（EEM）ポリシーを作成する方法について説明します。

この項には次のタスクが含まれます。

EEM Tcl スクリプトの登録と定義

環境変数を設定し、EEM ポリシーを登録するには、この作業を実行します。 EEM は、ポリシーそのものに含まれるイベント仕様に基づいてポリシーをスケジューリングし、実行します。EEM

ポリシーが登録されると、ソフトウェアによって、ポリシーが調べられ、指定されたイベントの発生時に実行されるよう、登録されます。

はじめる前に

Tcl スクリプト言語で作成されたポリシーが使用できる必要があります。 ポリシーの例については、EEM サンプル ポリシー, （32 ページ）を参照してください。 サンプル ポリシーがシステム ポリシー ディレクトリに格納されています。

手順の概要

1. **show event manager environment** [**all** | *environment-name*]
2. **configure**
3. **event manager environment** *var-name* [*var-value*]
4. ステップ 3, （24 ページ） を繰り返して、ステップ 5, （25 ページ） で登録されるポリシーに必要なすべての環境変数を設定します。
5. **event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]]
6. 次のいずれかのコマンドを使用してください。
   - **end**
   - **commit**

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**show event manager environment** [**all**	*environment-name*] 例： `RP/0/RSP0/CPU0:router# show event manager environment all`	（任意）EEM 環境変数の名前と値を表示します。 • **all** キーワードは、すべての EEM 環境変数を表示します。 • *environment-name* 引数は、指定された環境変数に関する情報を表示します。
ステップ 2	**configure** 例： `RP/0/RSP0/CPU0:router# configure`	グローバル コンフィギュレーション モードを開始します。
ステップ 3	**event manager environment** *var-name* [*var-value*] 例： `RP/0/RSP0/CPU0:router(config)# event manager environment _cron_entry 0-59/2 0-23/1 * * 0-7`	環境変数を新しい値にリセットします。 • *var-name* 引数は、EEM 環境設定変数に割り当てる名前です。 • *var-value* 引数は、環境変数 *var-name* に格納する、スペースを含む文字列です。

	コマンドまたはアクション	目的
		• 通例として、シスコで定義されているすべての環境変数の名前は、他の変数と区別するためにアンダースコア文字で始まります（_show_cmd など）。 • *var-value* 引数では、スペースを使用できます。 このコマンドは、*var-name* 引数の後から行の最後までのすべての文字列を *var-value* 引数の一部として解釈します。
ステップ 4	ステップ 3, （24 ページ） を繰り返して、ステップ 5, （25 ページ） で登録されるポリシーに必要なすべての環境変数を設定します。	—
ステップ 5	**event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]] 例： `RP/0/RSP0/CPU0:router(config)# event manager policy tm_cli_cmd.tcl username user_a type system`	ポリシー内で定義された指定イベントが発生した場合に、EEM ポリシーを実行するよう、定義します。 • シスコによって定義されたシステム ポリシーを登録するには、**system** キーワードを使用します。 • ユーザ定義のシステムポリシーを登録するには、**user** キーワードを使用します。 • ユーザ名認証が有効な期間を指定するには、**persist-time** キーワードを使用します。 この例では、tm_cli_cmd.tcl という名前の EEM サンプルポリシーが、システム ポリシーとして定義されます。

ステップ 6	次のいずれかのコマンドを使用してください。
	• **end**
	• **commit**
	
	例：
	`RP/0/RSP0/CPU0:router(config)# end`
	または
	`RP/0/RSP0/CPU0:router(config)# commit`
		設定変更を保存します。
		• **end** コマンドを実行すると、変更をコミットするように要求されます。
		`Uncommitted changes found, commit them before exiting(yes/no/cancel)? [cancel]:`
		◦ yes と入力すると、実行コンフィギュレーションファイルに変更が保存され、コンフィギュレーション セッションが終了して、ルータが EXEC モードに戻ります。
		◦ no と入力すると、コンフィギュレーションセッションが終了して、ルータが EXEC モードに戻ります。変更はコミットされません。
		◦ cancel と入力すると、現在のコンフィギュレーションセッションが継続します。コンフィギュ

	コマンドまたはアクション	目的
		レーションセッションは終了せず、設定変更もコミットされません。 • 実行コンフィギュレーション ファイルに設定変更を保存し、コンフィギュレーション セッションを継続するには、**commit** コマンドを使用します。

登録済みの EEM ポリシーの表示

登録済みの EEM ポリシーを表示するには、次の任意の作業を実行します。

手順の概要

1. **show event manager policy registered** [**event-type** *type*] [**system** | **user**] [**time-ordered** | **name-ordered**]

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**show event manager policy registered** [**event-type** *type*] [**system** | **user**] [**time-ordered** | **name-ordered**] 例： `RP/0/RSP0/CPU0:router# show event manager policy registered system`	現在登録されているポリシーについての情報を表示します。 • **event-type** キーワードを指定すると、特定のイベント タイプで登録されているポリシーが表示されます。 • **time-ordered** キーワードを指定すると、現在登録されているポリシーの情報が、時間でソートされて表示されます。 • **name-ordered** キーワードを指定すると、ポリシー名順（アルファベット順）にポリシーが表示されます。

EEM ポリシーの登録解除

EEM ポリシーを実行コンフィギュレーション ファイルから削除するには、次の作業を実行します。 ポリシーの実行はキャンセルされます。

手順の概要

1. **show event manager policy registered** [**event-type** *type*] [**system** | **user**] [**time-ordered** | **name-ordered**]
2. **configure**
3. **no event manager policy** *policy-name*
4. 次のいずれかのコマンドを使用してください。
   - **end**
   - **commit**
5. ステップ 1, （27 ページ） を繰り返して、ポリシーが削除されたことを確認します。

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**show event manager policy registered** [**event-type** *type*] [**system** | **user**] [**time-ordered** | **name-ordered**] 例： `RP/0/RSP0/CPU0:router# show event manager policy registered system`	現在登録されているポリシーについての情報を表示します。 • **event-type** キーワードを指定すると、特定のイベント タイプで登録されているポリシーが表示されます。 • **time-ordered** キーワードを指定すると、現在登録されているポリシーの情報が、時間でソートされて表示されます。 • **name-ordered** キーワードを指定すると、ポリシー名順（アルファベット順）にポリシーが表示されます。
ステップ 2	**configure** 例： `RP/0/RSP0/CPU0:router# configure`	グローバル コンフィギュレーション モードを開始します。
ステップ 3	**no event manager policy** *policy-name* 例： `RP/0/RSP0/CPU0:router(config)# no event manager policy tm_cli_cmd.tcl`	ポリシーを登録解除するために EEM ポリシーを設定から削除します。
ステップ 4	次のいずれかのコマンドを使用してください。 • **end** • **commit**	設定変更を保存します。 • **end** コマンドを実行すると、変更をコミットするように要求されます。 `Uncommitted changes found, commit them before exiting(yes/no/cancel)? [cancel]:`

	コマンドまたはアクション	目的

	例：
	`RP/0/RSP0/CPU0:router(config)# end`
	または
	`RP/0/RSP0/CPU0:router(config)# commit`
		◦ yes と入力すると、実行コンフィギュレーション ファイルに変更が保存され、コンフィギュレーションセッションが終了して、ルータが EXEC モードに戻ります。
		◦ no と入力すると、コンフィギュレーション セッションが終了して、ルータが EXEC モードに戻ります。変更はコミットされません。
		◦ cancel と入力すると、現在のコンフィギュレーションセッションが継続します。コンフィギュレーションセッションは終了せず、設定変更もコミットされません。
		• 実行コンフィギュレーション ファイルに設定変更を保存し、コンフィギュレーションセッションを継続するには、**commit** コマンドを使用します。

ステップ 5	ステップ 1, （27 ページ） を繰り返して、ポリシーが削除されたことを確認します。	—

EEM ポリシー実行の一時停止

すべての EEM ポリシーの実行をただちに一時停止するには、次の作業を実行します。 一時的なパフォーマンスまたはセキュリティ面での理由から、ポリシーの登録解除ではなく一時停止が必要なことがあります。

手順の概要

1. **show event manager policy registered** [**event-type** *type*] [**system** | **user**] [**time-ordered** | **name-ordered**]
2. **configure**
3. **event manager scheduler suspend**
4. 次のいずれかのコマンドを使用してください。
   - **end**
   - **commit**

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**show event manager policy registered** [**event-type** *type*] [**system** | **user**] [**time-ordered** | **name-ordered**] 例： `RP/0/RSP0/CPU0:router# show event manager policy registered system`	現在登録されているポリシーについての情報を表示します。 • **event-type** キーワードを指定すると、特定のイベント タイプで登録されているポリシーが表示されます。 • **time-ordered** キーワードを指定すると、現在登録されているポリシーの情報が、時間でソートされて表示されます。 • **name-ordered** キーワードを指定すると、ポリシー名順（アルファベット順）にポリシーが表示されます。
ステップ 2	**configure** 例： `RP/0/RSP0/CPU0:router# configure`	グローバル コンフィギュレーション モードを開始します。
ステップ 3	**event manager scheduler suspend** 例： `RP/0/RSP0/CPU0:router(config)# event manager scheduler suspend`	すべての EEM ポリシーの実行がすぐに一時停止されます。

ステップ 4	次のいずれかのコマンドを使用してください。
	• **end**
	• **commit**
	
	例：
	`RP/0/RSP0/CPU0:router(config)# end`
	または
	`RP/0/RSP0/CPU0:router(config)# commit`
		設定変更を保存します。
		• **end** コマンドを実行すると、変更をコミットするように要求されます。
		`Uncommitted changes found, commit them before exiting(yes/no/cancel)? [cancel]:`
		◦ yes と入力すると、実行コンフィギュレーション ファイルに変更が保存され、コンフィギュレーションセッションが終了して、ルータが EXEC モードに戻ります。
		◦ no と入力すると、コンフィギュレーションセッションが終了して、ルータが EXEC モードに戻ります。変更はコミットされません。
		◦ cancel と入力すると、現在のコンフィギュレーションセッションが継続します。コンフィギュレーションセッションは終了せず、設定変更もコミットされません。
		• 実行コンフィギュレーション ファイルに設定変更を保存し、コンフィギュレーションセッションを継続するには、**commit** コマンドを使用します。

	コマンドまたはアクション	目的

EEM ポリシーの管理

ユーザ ライブラリ ファイルまたはユーザ定義 EEM ポリシーの保存に使用するディレクトリを指定するには、この作業を実行します。

（注） この作業は、Tcl スクリプトを使用して記述される EEM ポリシーのみに適用されます。

手順の概要

1. **show event manager directory user** [**library** | **policy**]
2. **configure**
3. **event manager directory user** {**library** *path* | **policy** *path*}
4. 次のいずれかのコマンドを使用してください。
   - **end**
   - **commit**

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**show event manager directory user** [**library** | **policy**] 例： `RP/0/RSP0/CPU0:router# show event manager directory user library`	EEM ユーザ ライブラリまたはポリシー ファイルの保存に使用するディレクトリを表示します。 • オプションの **library** キーワードによって、ユーザ ライブラリ ファイルに使用されるディレクトリが表示されます。 • オプションの **policy** キーワードによって、ユーザ定義 EEM ポリシーに使用されるディレクトリが表示されます。
ステップ 2	**configure** 例： `RP/0/RSP0/CPU0:router# configure`	グローバル コンフィギュレーション モードを開始します。

	コマンドまたはアクション	目的
ステップ 3	**event manager directory user** {**library** *path* | **policy** *path*} 例： `RP/0/RSP0/CPU0:router(config)# event manager directory user library disk0:/usr/lib/tcl`	ユーザ ライブラリ ファイルまたはユーザ定義 EEM ポリシーの保存に使用するディレクトリを指定します。 • ユーザ ディレクトリへの絶対パス名を指定するには、*path* 引数を指定します。

ステップ 4	次のいずれかのコマンドを使用してください。
	
	• **end**
	
	• **commit**
	
	例：
	
	`RP/0/RSP0/CPU0:router(config)# end`
	または
	
	`RP/0/RSP0/CPU0:router(config)# commit`
		設定変更を保存します。
		
		• **end** コマンドを実行すると、変更をコミットするように要求されます。
		
		`Uncommitted changes found, commit them before exiting(yes/no/cancel)? [cancel]:`
		
		◦ yes と入力すると、実行コンフィギュレーション ファイルに変更が保存され、コンフィギュレーション セッションが終了して、ルータが EXEC モードに戻ります。
		
		◦ no と入力すると、コンフィギュレーション セッションが終了して、ルータが EXEC モードに戻ります。変更はコミットされません。
		
		◦ cancel と入力すると、現在のコンフィギュレーション セッションが継続します。コンフィギュレーション セッションは終了せず、設定変更もコミットされません。
		
		• 実行コンフィギュレーション ファイルに設定変更を保存し、コンフィギュレーション セッションを継続するには、**commit** コマンドを使用します。

EEM を使用したソフトウェア モジュール方式プロセスの信頼性メトリック

Cisco IOS XR ソフトウェア プロセスの信頼性メトリックを表示するには、この省略可能なタスクを実行します。

手順の概要

1. **show event manager metric process** {**all** | *job-id* | *process-name*} **location** {**all** | *node-id*}

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**show event manager metric process** {**all** | *job-id* | *process-name*} **location** {**all** | *node-id*} 例： `RP/0/RSP0/CPU0:router# show event manager environment`	プロセスの信頼性メトリック データを表示します。 システムでは、プロセスの開始時と終了時にレコードが保存され、このデータが、信頼性分析の基本データとして使用されます。

EEM サンプル ポリシーの変更

サンプル ポリシーの 1 つを変更するには、この作業を実行します。 Cisco IOS XR ソフトウェアには、EEM を含むいくつかのサンプル ポリシーがイメージに含まれています。 EEM ポリシーの開発者は、ポリシーが実行されるイベントと、イベントの記録および応答に関連付けられているオプションを、カスタマイズすることによって、これらのポリシーを変更できます。 さらに、開発者は、ポリシーの実行時に実装されるアクションを選択できます。

EEM サンプル ポリシー

Cisco IOS XR ソフトウェアには、サンプル ポリシーが含まれています（表「*EEM* サンプル ポリシーの説明」を参照してください）。 サンプル ポリシーは、ユーザ ディレクトリにコピーして変更できます。 Tcl は、現在ポリシー作成のためにシスコでサポートされている唯一のスクリプト言語です。 Tcl ポリシーは、Emacs などのテキスト エディタを使用して変更できます。 ポリシーは、定義されている経過時間の秒数以内で実行する必要があり、時間変数はポリシー内で設定できます。 デフォルトは 20 秒です。

この表ではサンプル EEM ポリシーについて説明します。

表 *7*：*EEM* サンプル ポリシーの説明

ポリシーの名前	説明
periodic_diag_cmds.tcl	このポリシーは、cron エントリ _cron_entry_diag の期限が切れたときにトリガーされます。その後、この固定セットの出力が固定のコマンド セットに対して収集され、出力が電子メールで送信されます。

ポリシーの名前	説明
periodic_proc_avail.tcl	このポリシーは、cron エントリ _cron_entry_procavail の期限が切れたときにトリガーされます。 その後、この固定セットの出力が固定のコマンドセットに対して収集され、出力が電子メールで送信されます。
periodic_sh_log.tcl	このポリシーは、cron エントリ _cron_entry_log の期限が切れたときにトリガーされ、show log コマンドとその他いくつかのコマンドの出力が収集されます。 環境変数 _log_past_hours が設定されている場合、最後の _log_past_hours 時間に生成されたログ メッセージが収集されます。 そうでない場合、ログ全体が収集されます。
sl_sysdb_timeout.tcl	このポリシーは、スクリプトが sysdb タイムアウト ios_msgs を探すときにトリガーされ、show コマンドの出力が取得されます。 出力は、ブロックしているプロセスの名前が付けられたファイルに書き込まれます。
tm_cli_cmd.tcl	このポリシーは、設定可能な CRON エントリを使用して実行されます。 設定可能な CLI コマンドが実行され、結果が電子メールで送信されます。
tm_crash_hist.tcl	このポリシーは、毎日夜中に実行され、指定された電子メールアドレスにプロセス クラッシュ履歴レポートが電子メールで送信されます。

使用可能なサンプル ポリシーおよびその実行方法についての詳細は、EEM イベント ディテクタのデモ：例 , （55 ページ）を参照してください。

手順の概要

1. **show event manager policy available** [**system** | **user**]
2. 画面に表示されたサンプル ポリシーの内容を、テキスト エディタにカット アンド ペーストします。
3. ポリシーを編集し、新しいファイル名で保存します。
4. 新しいファイルを、ルータのフラッシュ メモリにコピーして戻します。
5. **configure**
6. **event manager directory user** {**library** *path* | **policy** *path*}
7. **event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]]
8. 次のいずれかのコマンドを使用してください。
   - **end**
   - **commit**

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**show event manager policy available** [**system** | **user**] 例： `RP/0/RSP0/CPU0:router# show event manager policy available`	登録可能な EEM ポリシーを表示します。
ステップ 2	画面に表示されたサンプルポリシーの内容を、テキスト エディタにカット アンド ペーストします。	—
ステップ 3	ポリシーを編集し、新しいファイル名で保存します。	—
ステップ 4	新しいファイルを、ルータのフラッシュ メモリにコピーして戻します。	—
ステップ 5	**configure** 例： `RP/0/RSP0/CPU0:router# configure`	グローバル コンフィギュレーション モードを開始します。

	コマンドまたはアクション	目的
ステップ 6	**event manager directory user** {**library** *path* | **policy** *path*} 例： `RP/0/RSP0/CPU0:router(config)# event manager directory user library disk0:/user_library`	ユーザ ライブラリ ファイルまたはユーザ定義 EEM ポリシーの保存に使用するディレクトリを指定します。
ステップ 7	**event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]] 例： `RP/0/RSP0/CPU0:router(config)# event manager policy test.tcl username user_a type user`	ポリシー内で定義された指定イベントが発生した場合に、EEM ポリシーを実行するよう、定義します。

ステップ 8	次のいずれかのコマンドを使用してください。
	• **end**
	• **commit**
	
	例：
	`RP/0/RSP0/CPU0:router(config)# end`
	または
	`RP/0/RSP0/CPU0:router(config)# commit`
		設定変更を保存します。
		• **end** コマンドを実行すると、変更をコミットするように要求されます。
		`Uncommitted changes found, commit them before exiting(yes/no/cancel)? [cancel]:`
		◦ yes と入力すると、実行コンフィギュレーションファイルに変更が保存され、コンフィギュレーションセッションが終了して、ルータが EXEC モードに戻ります。
		◦ no と入力すると、コンフィギュレーションセッションが終了して、ルータが EXEC モードに戻ります。変更はコミットされません。
		◦ cancel と入力すると、現在のコンフィギュレーション セッションが継続します。コンフィギュレーションセッションは終了せず、設定変更もコミットされません。
		• 実行コンフィギュレーションファイルに設定変更を保存し、コンフィギュレーションセッションを継続するには、**commit** コマンドを使用します。

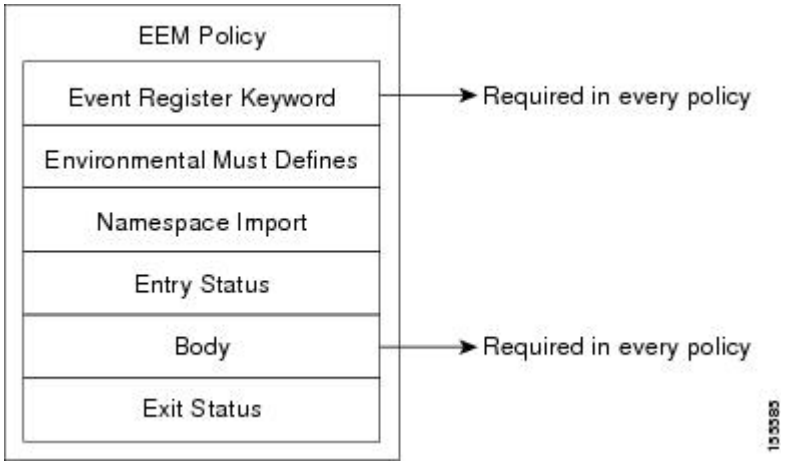
Tcl を使用した EEM ポリシーのプログラミング

Tcl コマンド拡張を使用してポリシーをプログラムするには、この作業を実行します。 既存のポリシーをコピーし、変更することを推奨します。 EEM Tcl ポリシーには、event_register Tcl コマンド拡張と本体の 2 つの必須部分が存在する必要があります。 Tcl ポリシーの構造と要件, （36 ページ） にある他のすべてのセクションは、オプションです。

Tcl ポリシーの構造と要件

すべての EEM ポリシーでは、図 1：Tcl ポリシーの構造と要件，（36 ページ） に示されているように、同じ構造が共有されます。 EEM ポリシーには、event_register Tcl コマンド拡張と本体の、2 つの必須部分が存在します。 ポリシーの残りの部分の、環境定義必須、名前空間のインポート、開始ステータス、および終了ステータスは、オプションです。

図 *1*：*Tcl* ポリシーの構造と要件



各ポリシーの開始は、**event_register** Tcl コマンド拡張を使用して検出するために、イベントを示し、登録する必要があります。 ポリシーのこの部分によって、ポリシーの実行がスケジュールされます。 使用可能な EEM **event_register** Tcl コマンド拡張のリストについては、Embedded Event Manager イベント登録 Tcl コマンド拡張, （68 ページ）を参照してください。 次に、**event_register_timer** Tcl コマンド拡張を登録する Tcl コード例を示します。

```
::cisco::eem::event_register_timer cron name crontimer2 cron_entry $_cron_entry maxrun 240
```

次に、一部の環境変数をチェックし、定義する Tcl コードの例を示します。

```
# Check if all the env variables that we need exist.
# If any of them does not exist, print out an error msg and quit.
if {![info exists _email_server]} {
  set result \
    "Policy cannot be run: variable _email_server has not been set"
  error $result $errorInfo
}
if {![info exists _email_from]} {
  set result \
    "Policy cannot be run: variable _email_from has not been set"
  error $result $errorInfo
```

```
}
if {![info exists _email_to]} {
  set result \
    "Policy cannot be run: variable _email_to has not been set"
  error $result $errorInfo
}
```

名前空間のインポート セクションはオプションで、コード ライブラリが定義されます。 次に、名前空間インポート セクションを設定する Tcl コードの例を示します。

```
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
```

ポリシーの本体は必須の構造で、次のものを含める必要があります。

- 検出されたイベントに関する情報の EEM への問い合わせに使用される **event_reqinfo** イベント情報の Tcl コマンド拡張。 使用可能な EEM イベント情報の Tcl コマンド拡張のリストについては、Embedded Event Manager イベント情報 Tcl コマンド拡張, （96 ページ）を参照してください。
- EEM 特有のアクションの指定に使用される、**action_syslog** などのアクション Tcl コマンド拡張。 使用可能な EEM アクションの Tcl コマンド拡張のリストについては、Embedded Event Manager アクション Tcl コマンド拡張, （117 ページ）を参照してください。
- 一般的なシステム情報の取得に使用される、**sys_reqinfo_routername** などのシステム情報の Tcl コマンド拡張。 使用可能な EEM システム情報の Tcl コマンド拡張のリストについては、Embedded Event Manager システム情報 Tcl コマンド拡張, （135 ページ）を参照してください。
- ポリシーからの、SMTP ライブラリ（電子メール通知を送信）または CLI ライブラリ（CLI コマンドを実行）の使用。 使用可能な SMTP ライブラリの Tcl コマンド拡張のリストについては、SMTP ライブラリのコマンド拡張, （147 ページ）を参照してください。 使用可能な CLI ライブラリの Tcl コマンド拡張のリストについては、CLI ライブラリのコマンド拡張, （149 ページ）を参照してください。
- 他のポリシーによって使用される Tcl 変数の保存に使用される **context_save** および **context_retrieve** の Tcl コマンド拡張。

次に、イベントを問い合わせ、本体セクションの一部としてメッセージを記録するコードの Tcl コードの例を示します。

```
# Query the event info and log a message.
array set arr_einfo [event_reqinfo]
if {$_cerrno != 0} {
  set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
    $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
  error $result
}
global timer_type timer_time_sec
set timer_type $arr_einfo(timer_type)
  set timer_time_sec $arr_einfo(timer_time_sec)
# Log a message.
set msg [format "timer event: timer type %s, time expired %s" \
$timer_type [clock format $timer_time_sec]]
action_syslog priority info msg $msg
if {$_cerrno != 0} {
  set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
    $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
  error $result
}
```

EEM 開始ステータス

EEMポリシーの開始ステータスの部分は、前のポリシーが同じイベントに対して実行されたかどうかや、前のポリシーの終了ステータスを特定するために、使用されます。 _entry_status 変数が定義されている場合、このイベントに対して前のポリシーがすでに実行されています。 _entry_status 変数の値によって、前のポリシーの戻りコードが特定されます。

エントリ ステータス指定は、次の 3 つのうちのいずれかの値を使用します。

- 0（以前のポリシーは成功した）
- 0 以外（以前のポリシーは失敗した）
- Undefined（以前実行されたポリシーはない）

EEM 終了ステータス

ポリシーでそのコードの実行を終了すると、終了値が設定されます。 終了値は、EEMによって使用され、このイベントのデフォルト アクションがある場合に、それが適用されたかどうかが判断されます。 値 0 は、デフォルトのアクションを実行しないことを意味します。 0 以外の値は、デフォルトのアクションを実行する必要があることを意味します。 終了ステータスは、同じイベントで実行される後続ポリシーに渡されます。

EEM ポリシーと Cisco エラー番号

一部の EEM Tcl コマンド拡張によって、Cisco エラー番号の Tcl グローバル変数の _cerrno が設定されます。 _cerrno が設定されるたびに、他の 4 つの Tcl グローバル変数が _cerrno から分岐し、それとともに設定されます（_cerr_sub_num、_cerr_sub_err、_cerr_posix_err、および _cerr_str）。

たとえば、次の例の **action_syslog** コマンドでは、コマンド実行の副次的な影響としてこれらのグローバル変数が設定されます。

```
action_syslog priority warning msg "A sample message generated by action_syslog"
if {$_cerrno != 0} {
  set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
    $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
  error $result
}
```

_cerrno：32 ビット エラー戻り値

コマンドによって設定された _cerrno は、次の形式の 32 ビットの整数を表す場合があります。

```
XYSSSSSSSSSSSSSEEEEEEEEPPPPPPPPP
```

たとえば、次のエラー戻り値は、EEM Tcl コマンド拡張から戻される場合があります。

```
862439AE
```

この番号は、次の 32 ビット値として解釈されます。

```
10000110001001000011100110101110
```

この 32 ビットの整数は、次の表に示されているように、5 つの変数に分けられます。

表 *8*：*_cerrno*：*32* ビット エラー戻り値の変数

変数	説明
XY	エラー クラス（エラーの重大度を示します）。この変数は、32 ビットのエラー戻り値の最初の 2 ビットに対応しています。前述のケースの 10 は、CERR_CLASS_WARNING を示します。 この変数特有の 4 つのエラー クラス エンコーディングについては、表 9：エラー クラス エンコーディング，（39 ページ） を参照してください。
SSSSSSSSSSSSSS	最新のエラーが生成されたサブシステム番号（13 ビット = 値 8192）。 これは、32 ビット シーケンスの次の 13 ビットで、その整数値は $_cerr_sub_num に含まれています。
EEEEEEEE	サブシステム固有のエラー番号（8 ビット = 値 256）。 このセグメントは、32 ビット シーケンスの次の 8 ビットで、このエラー番号に対応する文字列は、$_cerr_sub_err に含まれています。
PPPPPPPP	パススルー POSIX エラー コード（9 ビット = 値 512）。 これは、32 ビット シーケンスの最後で、このエラー コードに対応する文字列は、$_cerr_posix_err に含まれています。

XY のエラー クラス エンコーディング

最初の変数 XY は、次の表に示されているように、エラー クラス エンコーディングを参照しています。

表 *9*：エラー クラス エンコーディング

エラー戻り値	エラー クラス
00	CERR_CLASS_SUCCESS
01	CERR_CLASS_INFO

エラー戻り値	エラー クラス
10	CERR_CLASS_WARNING
11	CERR_CLASS_FATAL

ゼロのエラー戻り値は、SUCCESS を示します。

手順の概要

1. **show event manager policy available** [**system** | **user**]
2. 画面に表示されたサンプル ポリシーの内容を、テキスト エディタにカット アンド ペーストします。
3. 必要な event_register Tcl コマンド拡張を定義します。
4. 適切な名前空間を、::cisco 階層構造に追加します。
5. Must Define セクションをプログラムし、このポリシーで使用される各環境変数をチェックします。
6. スクリプトの本体をプログラムします。
7. 開始ステータスをチェックし、ポリシーがこのイベントに対して前に実行されたかどうかを判断します。
8. 終了ステータスをチェックし、デフォルト アクションが存在する場合に、このイベントのデフォルト アクションが適用されたかどうかを判断します。
9. Cisco エラー番号（_cerrno）の Tcl グローバル変数を設定します。
10. 新しいファイル名で Tcl スクリプトを保存し、Tcl スクリプトをルータにコピーします。
11. **configure**
12. **event manager directory user** {**library** *path* | **policy** *path*}
13. **event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]]
14. 次のいずれかのコマンドを使用してください。
    - **end**
    - **commit**
15. ポリシーを実行し、ポリシーを観察します。
16. ポリシーが正しく実行されていない場合、デバッグのテクニックを使用します。

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**show event manager policy available** [**system** | **user**]	登録可能な EEM ポリシーを表示します。

	コマンドまたはアクション	目的
	例： `RP/0/RSP0/CPU0:router# show event manager policy available`	
ステップ 2	画面に表示されたサンプル ポリシーの内容を、テキスト エディタにカット アンド ペーストします。	—
ステップ 3	必要な event_register Tcl コマンド拡張を定義します。	検出するイベントについて、適切な event_register Tcl コマンド拡張を選択し、ポリシーに追加します。 有効なイベント登録 Tcl コマンド拡張を次に示します。 • event_register_appl • event_register_counter • event_register_stat • event_register_wdsysmon • event_register_oir • event_register_process • event_register_syslog • event_register_timer • event_register_timer_subscriber • event_register_hardware • event_register_none
ステップ 4	適切な名前空間を、::cisco 階層構造に追加します。	ポリシーの開発者は、Cisco IOS XR EEM によって使用されるすべての拡張をグループ化するため、Tcl ポリシーで新しい名前空間 ::cisco を使用できます。 ::cisco 階層の下には、2 つの名前空間があります。各名前空間に属する名前空間と EEM Tcl コマンド拡張カテゴリは次のとおりです。 • ::cisco::eem ◦ EEM イベント登録 ◦ EEM イベント情報 ◦ EEM イベント パブリッシュ ◦ EEM アクション ◦ EEM ユーティリティ ◦ EEM コンテキスト ライブラリ

	コマンドまたはアクション	目的
		◦ EEM システム情報 ◦ CLI ライブラリ • ::cisco::lib ◦ SMTP ライブラリ （注） 適切な名前空間がインポートされていることを確認するか、前述のコマンドを使用する場合は修飾されたコマンド名を使用します。
ステップ 5	Must Define セクションをプログラムし、このポリシーで使用される各環境変数をチェックします。	この手順は任意です。 Must Define は、ポリシーによって必要とされるすべての EEM 環境変数が、回復アクションの実行前に定義されているかどうかをテストする、ポリシーのセクションです。 ポリシーによって EEM 環境変数が使用されない場合、Must Define セクションは不要です。 EEM スクリプトの EEM 環境変数は、ポリシーの実行前にポリシーに対して外部定義された Tcl グローバル変数です。 EEM 環境変数を定義するには、EEM コンフィギュレーション コマンド **event manager environment** を使用します。 規則として、すべてのシスコ EEM 環境変数の先頭は、「_」（アンダースコア）になっています。 将来的な競合を避けるため、「_」で始まる新しい変数を定義しないことを推奨します。 （注） EXEC モードで **show event manager environment** コマンドを使用して、システムの Embedded Event Manager 環境変数セットを表示できます。 たとえば、サンプル ポリシーで定義されている EEM 環境変数には、電子メール変数が含まれます。 適切に動作させるためには、電子メールを送信するサンプル ポリシーに、次に示す変数が設定されている必要があります。 EEM サンプル ポリシーで使用される電子メール特有の環境変数について説明します。 • **_email_server**：電子メール送信に使用されるシンプル メール転送プロトコル（SMTP）メール サーバ（たとえば mailserver.example.com） • **_email_to**：電子メールの送信先アドレス（たとえば engineering@example.com） • **_email_from**：電子メールの送信元アドレス（たとえば devtest@example.com） • **_email_cc**：電子メールのコピーの送信先アドレス（たとえば manager@example.com）

	コマンドまたはアクション	目的
ステップ 6	スクリプトの本体をプログラムします。	スクリプトのこのセクションでは、次のいずれかを定義できます。 • 検出されたイベントに関する情報の EEM への問い合わせに使用される **event_reqinfo** イベント情報の Tcl コマンド拡張。 • EEM 特有のアクションの指定に使用される、**action_syslog** などのアクション Tcl コマンド拡張。 • 一般的なシステム情報の取得に使用される、**sys_reqinfo_routername** などのシステム情報の Tcl コマンド拡張。 • 他のポリシーによって使用される Tcl 変数の保存に使用される **context_save** および **context_retrieve** の Tcl コマンド拡張。 • ポリシーからの、SMTP ライブラリ（電子メール通知を送信）または CLI ライブラリ（CLI コマンドを実行）の使用。
ステップ 7	開始ステータスをチェックし、ポリシーがこのイベントに対して前に実行されたかどうかを判断します。	前のポリシーが正常終了した場合、現在のポリシーは、実行が必要な場合と、実行が不要な場合があります。 開始ステータス指定には、0（前のポリシーが正常終了した）、Not=0（前のポリシーに障害が発生した）、および Undefined（実行された前のポリシーがない）の、3 つの値のうちいずれか 1 つを使用できます。
ステップ 8	終了ステータスをチェックし、デフォルトアクションが存在する場合に、このイベントのデフォルトアクションが適用されたかどうかを判断します。	値 0 は、デフォルトのアクションを実行しないことを意味します。 0 以外の値は、デフォルトのアクションを実行する必要があることを意味します。 終了ステータスは、同じイベントで実行される後続ポリシーに渡されます。
ステップ 9	Cisco エラー番号（_cerrno）の Tcl グローバル変数を設定します。	一部の EEM Tcl コマンド拡張によって、Cisco エラー番号の Tcl グローバル変数の _cerrno が設定されます。 _cerrno が設定されるたびに、他の 4 つの Tcl グローバル変数が _cerrno から分岐し、それとともに設定されます（_cerr_sub_num、_cerr_sub_err、_cerr_posix_err、および _cerr_str）。
ステップ 10	新しいファイル名で Tcl スクリプトを保存し、Tcl スクリプトをルータにコピーします。	Embedded Event Manager ポリシー ファイル名は、次の仕様に従っています。 • オプションのプレフィックス Mandatory. がある場合、これは、システム ポリシーがまだ登録されていない場合に、自動的に登録される必要があるシステム ポリシーであることを示します。 たとえば、Mandatory.sl_text.tcl などです。 • 指定された 1 つめのイベントの 2 文字の省略形が含まれるファイル名の本体部（表 2：2 文字の省略形の指定，（9 ページ） を参照）、下線文字部、および、ポリシーをさらに示す説明フィールド部。

	コマンドまたはアクション	目的
		• ファイル名拡張子部は .tcl と定義されます。 詳細については、Embedded Event Manager 用のシスコ ファイル命名規則, （8 ページ）を参照してください。 ルータ上のフラッシュ ファイル システム（通常は disk0:）に、ファイルをコピーします。
ステップ 11	**configure** 例： `RP/0/RSP0/CPU0:router# configure`	グローバル コンフィギュレーション モードを開始します。

ステップ 12	**event manager directory user** {**library** *path* | **policy** *path*}
	
	例：
	`RP/0/RSP0/CPU0:router(config)# event manager directory user library`
	
	`disk0:/user_library`
		ユーザ ライブラリ ファイルまたはユーザ定義 EEM ポリシーの保存に使用するディレクトリを指定します。

ステップ 13	**event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]] 例： `RP/0/RSP0/CPU0:router(config)# event manager policy test.tcl username user_a type user`	ポリシー内で定義された指定イベントが発生した場合に、EEM ポリシーを実行するよう、定義します。

ステップ 14	次のいずれかのコマンドを使用してください。
	• **end**
	• **commit**
	
	例：
	
	`RP/0/RSP0/CPU0:router(config)# end`
	または
	
	`RP/0/RSP0/CPU0:router(config)# commit`
		設定変更を保存します。
		• **end** コマンドを実行すると、変更をコミットするように要求されます。
		
		`Uncommitted changes found, commit them before exiting(yes/no/cancel)? [cancel]:`
		
		◦ yes と入力すると、実行コンフィギュレーション ファイルに変更が保存され、コンフィギュレーション セッションが終了して、ルータが EXEC モードに戻ります。
		
		◦ no と入力すると、コンフィギュレーション セッションが終了して、ルータが EXEC モードに戻ります。変更はコミットされません。

	コマンドまたはアクション	目的
		◦ cancel と入力すると、現在のコンフィギュレーション セッションが継続します。コンフィギュレーション セッションは終了せず、設定変更もコミットされません。 • 実行コンフィギュレーション ファイルに設定変更を保存し、コンフィギュレーション セッションを継続するには、**commit** コマンドを使用します。
ステップ 15	ポリシーを実行し、ポリシーを観察します。	—
ステップ 16	ポリシーが正しく実行されていない場合、デバッグのテクニックを使用します。	—

EEM ユーザ Tcl ライブラリ索引の作成

Tcl ファイルのライブラリに含まれているすべての手順のディレクトリが含まれている、索引ファイルを作成するには、この作業を実行します。 この作業を行うと、EEM Tcl のライブラリ サポートをテストできます。 この作業では、Tcl ライブラリ ファイルが含まれるライブラリ ディレクトリが作成され、ファイルがディレクトリにコピーされ、ライブラリ ファイルにあるすべての手順のディレクトリが含まれる索引 tclIndex が作成されます。 索引が作成されない場合、Tcl 手順を参照する EEM ポリシーを実行するときに、Tcl 手順は見つかりません。

手順の概要

1. ワークステーション（UNIX、Linux、PC、または Mac）で、ライブラリ ディレクトリを作成し、Tcl ライブラリ ファイルをディレクトリにコピーします。
2. **tclsh**
3. **auto_mkindex** *directory_name* ***.tcl**
4. ターゲット ルータ上のユーザ ライブラリ ファイルの保存に使用されるディレクトリに、ステップ 1, （46 ページ） から Tcl ライブラリ ファイルをコピーし、ステップ 3, （47 ページ） から tclIndex ファイルをコピーします。
5. Tcl で記述されたユーザ定義 EEM ポリシー ファイルを、ターゲット ルータ上でユーザ定義 EEM ポリシーの保存に使用されるディレクトリにコピーします。
6. **configure**
7. **event manager directory user library** *path*
8. **event manager directory user policy** *path*
9. **event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]]
10. **event manager run** *policy* [*argument*]
11. 次のいずれかのコマンドを使用してください。
    - **end**
    - **commit**

手順の詳細

	コマンドまたはアクション	目的

ステップ 1	ワークステーション（UNIX、Linux、PC、または Mac）で、ライブラリ ディレクトリを作成し、Tcl ライブラリ ファイルをディレクトリにコピーします。	次の例ファイルを使用すると、Tcl シェルが実行されているワークステーション上で、tclIndex を作成できます。
		lib1.tcl
		`proc test1 {} {`
		`  puts "In procedure test1"`
		`}`
		`proc test2 {} {`
		`  puts "In procedure test2"`
		`}`
		lib2.tcl
		`proc test3 {} {`
		`  puts "In procedure test3"`
		`}`

	コマンドまたはアクション	目的
ステップ 2	**tclsh** 例： `workstation% tclsh`	Tcl シェルを開始します。

ステップ 3	**auto_mkindex** *directory_name* ***.tcl**
	
	例：
	`workstation% auto_mkindex eem_library *.tcl`
		auto_mkindex コマンドを使用して、tclIndex ファイルを作成します。 すべての手順のディレクトリが含まれる tclIndex ファイルは、Tcl ライブラリ ファイルに含まれていました。 どのディレクトリにも 1 つの tclIndex ファイルのみを存在させることができ、他の Tcl ファイルはグループ化しておくことが可能であるため、ディレクトリ内で **auto_mkindex** を実行することを推奨します。 ディレクトリ内で **auto_mkindex** を実行すると、特定の tclIndex を使用してどの Tcl ソース ファイルを索引化できるかが判断されます。
		
		lib1.tcl ファイルと lib2.tcl ファイルがライブラリ ファイル ディレクトリにあり、**auto_mkindex** コマンドが実行されたときに、次の例に示す tclIndex が作成されます。
		
		tclIndex
		
		`# Tcl autoload index file, version 2.0`
		`# This file is generated by the "auto_mkindex" command`
		`# and sourced to set up indexing information for one or`
		`# more commands.  Typically each line is a command that`
		`# sets an element in the auto_index array, where the`
		`# element name is the name of a command and the value is`
		`# a script that loads the command.`
		`set auto_index(test1) [list source [file join $dir lib1.tcl]]`
		`set auto_index(test2) [list source [file join $dir lib1.tcl]]`
		`set auto_index(test3) [list source [file join $dir lib2.tcl]]`

ステップ 4	ターゲット ルータ上のユーザ ライブラリ ファイルの保存に使用されるディレクトリに、ステップ 1, （46 ページ）から Tcl ライブラリ ファイルをコピーし、ステップ 3, （47 ページ） から tclIndex ファイルをコピーします。	—
ステップ 5	Tcl で記述されたユーザ定義 EEM ポリシー ファイルを、ターゲット ルータ上でユーザ定義 EEM ポリシーの保存に使用されるディレクトリにコピーします。	ディレクトリは、ステップ 4, （47 ページ） で使用されるディレクトリと同じディレクトリを使用できます。 次に、EEM でサポートされる Tcl ライブラリのテストに、ユーザ定義 EEM ポリシーを使用できる例を示します。

	コマンドまたはアクション	目的

		libtest.tcl
		
		`::cisco::eem::event_register_none`
		`namespace import ::cisco::eem::*`
		`namespace import ::cisco::lib::*`
		`global auto_index auto_path`
		`puts [array names auto_index]`
		`if { [catch {test1} result]} {`
		`  puts "calling test1 failed result = $result $auto_path"`
		`}`
		`if { [catch {test2} result]} {`
		`  puts "calling test2 failed result = $result $auto_path"`
		`}`
		`if { [catch {test3} result]} {`
		`  puts "calling test3 failed result = $result $auto_path"`
		`}`

ステップ 6	**configure** 例： `RP/0/RSP0/CPU0:router# configure`	グローバル コンフィギュレーション モードを開始します。
ステップ 7	**event manager directory user library** *path* 例： `RP/0/RSP0/CPU0:router(config)# event manager directory user library disk2:/eem_library`	EEMユーザ ライブラリのディレクトリを指定します。これは、ステップ 4, （47 ページ）のファイルがコピーされたディレクトリです。
ステップ 8	**event manager directory user policy** *path* 例： `RP/0/RSP0/CPU0:router(config)# event manager directory user policy disk2:/eem_policies`	EEM ユーザ ポリシーのディレクトリを指定します。これは、ステップ 5, （47 ページ）のファイルがコピーされたディレクトリです。
ステップ 9	**event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]] 例： `RP/0/RSP0/CPU0:router(config)# event manager policy libtest.tcl username user_a`	ユーザ定義の EEM ポリシーを登録します。
ステップ 10	**event manager run** *policy* [*argument*] 例： `RP/0/RSP0/CPU0:router(config)# event manager run libtest.tcl`	手動で EEM ポリシーを実行します。

	コマンドまたはアクション	目的

ステップ 11	次のいずれかのコマンドを使用してください。
	• **end**
	• **commit**
	
	例：
	`RP/0/RSP0/CPU0:router(config)# end`
	または
	`RP/0/RSP0/CPU0:router(config)# commit`
		設定変更を保存します。
		• **end** コマンドを実行すると、変更をコミットするように要求されます。
		`Uncommitted changes found, commit them before exiting(yes/no/cancel)? [cancel]:`
		◦ yes と入力すると、実行コンフィギュレーションファイルに変更が保存され、コンフィギュレーションセッションが終了して、ルータが EXEC モードに戻ります。
		◦ no と入力すると、コンフィギュレーション セッションが終了して、ルータが EXEC モードに戻ります。変更はコミットされません。
		◦ cancel と入力すると、現在のコンフィギュレーションセッションが継続します。コンフィギュレーションセッションは終了せず、設定変更もコミットされません。
		• 実行コンフィギュレーション ファイルに設定変更を保存し、コンフィギュレーションセッションを継続するには、**commit** コマンドを使用します。

EEM ユーザ Tcl パッケージ索引の作成

すべての Tcl パッケージのディレクトリと、Tcl パッケージ ファイルのライブラリに含まれるバージョン情報が含まれる、Tcl パッケージの索引ファイルを作成するには、この作業を実行します。Tcl パッケージは、Tcl **package** キーワードを使用してサポートされます。

Tcl パッケージは、EEM システム ライブラリ ディレクトリまたは EEM ユーザ ライブラリ ディレクトリのいずれかにあります。 **package require** Tcl コマンドが実行されると、ユーザ ライブラリ ディレクトリで、まず、pkgIndex.tcl ファイルが検索されます。 pkgIndex.tcl ファイルがユーザ ディレクトリで見つからない場合、システム ライブラリ ディレクトリが検索されます。

この作業では、**pkg_mkIndex** コマンドを使用して、適切なライブラリ ディレクトリに Tcl パッケージディレクトリ pkgIndex.tcl ファイルが作成され、バージョン情報とともに、ディレクトリに含まれるすべての Tcl パッケージについての情報が含められます。 索引が作成されない場合、**package require** Tcl コマンドが含まれる EEM ポリシーが実行されたときに、Tcl パッケージは見つかりません。

EEM で Tcl パッケージ サポートを使用すると、ユーザは、Tcl の XML_RPC などのパッケージにアクセスできます。 Tcl パッケージインデックスが作成されるとき、Tcl スクリプトは、外部エンティティに対する XML-RPC 呼び出しを容易に行うことができます。

（注） C プログラミング コードで実装されるパッケージは、EEM ではサポートされません。

手順の概要

1. ワークステーション（UNIX、Linux、PC、または Mac）で、ライブラリ ディレクトリを作成し、Tcl パッケージ ファイルをディレクトリにコピーします。
2. **tclsh**
3. **pkg_mkindex** *directory_name* ***.tcl**
4. ターゲット ルータ上のユーザ ライブラリ ファイルの保存に使用されるディレクトリに、ステップ 1, （50 ページ） から Tcl パッケージ ファイルをコピーし、ステップ 3, （51 ページ） から pkgIndex ファイルをコピーします。
5. Tcl で記述されたユーザ定義 EEM ポリシー ファイルを、ターゲット ルータ上でユーザ定義 EEM ポリシーの保存に使用されるディレクトリにコピーします。
6. **configure**
7. **event manager directory user library** *path*
8. **event manager directory user policy** *path*
9. **event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]]
10. **event manager run** *policy* [*argument*]
11. 次のいずれかのコマンドを使用してください。
    - **end**
    - **commit**

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	ワークステーション（UNIX、Linux、PC、または Mac）で、ライブラリ ディレクトリを作成し、Tcl パッケージ ファイルをディレクトリにコピーします。	–
ステップ 2	**tclsh** 例： `workstation% tclsh`	Tcl シェルを開始します。

	コマンドまたはアクション	目的

ステップ 3	**pkg_mkindex** *directory_name* ***.tcl**
	
	例：
	`workstation% pkg_mkindex eem_library *.tcl`
		pkg_mkindex コマンドを使用して、pkgIndex ファイルを作成します。 すべてのパッケージのディレクトリが含まれる pkgIndex ファイルは、Tcl ライブラリ ファイルに含まれていました。 どのディレクトリにも 1 つの pkgIndex ファイルのみを存在させることができ、他の Tcl ファイルはグループ化しておくことが可能であるため、ディレクトリ内で **pkg_mkindex** コマンドを実行することを推奨します。 ディレクトリ内で **pkg_mkindex** コマンドを実行すると、特定の pkgIndex を使用してどの Tcl パッケージ ファイルを索引化できるかが判断されます。
		
		次に、いくつかの Tcl パッケージがライブラリ ファイル ディレクトリにあり、pkg_mkindex コマンドが実行されたときに、pkgIndex が作成される例を示します。
		
		pkgIndex
		
		`# Tcl package index file, version 1.1`
		`# This file is generated by the "pkg_mkIndex" command`
		`# and sourced either when an application starts up or`
		`# by a "package unknown" script. It invokes the`
		`# "package ifneeded" command to set up package-related`
		`# information so that packages will be loaded automatically`
		`# in response to "package require" commands. When this`
		`# script is sourced, the variable $dir must contain the`
		`# full path name of this file's directory.`
		`package ifneeded xmlrpc 0.3 [list source [file join $dir xmlrpc.tcl]]`

ステップ 4	ターゲット ルータ上のユーザ ライブラリ ファイルの保存に使用されるディレクトリに、ステップ 1, （50 ページ） から Tcl パッケージ ファイルをコピーし、ステップ 3, （51 ページ） から pkgIndex ファイルをコピーします。	—

ステップ 5	Tcl で記述されたユーザ定義 EEM ポリシー ファイルを、ターゲット ルータ上でユーザ定義 EEM ポリシーの保存に使用されるディレクトリにコピーします。	ディレクトリは、ステップ 4, （51 ページ） で使用されるディレクトリと同じディレクトリを使用できます。
		
		次に、EEM でサポートされる Tcl ライブラリのテストに、ユーザ定義 EEM ポリシーを使用できる例を示します。
		
		packagetest.tcl
		
		`::cisco::eem::event_register_none maxrun 1000000.000`
		`#`
		`# test if xmlrpc available`
		`#`
		`#`

	コマンドまたはアクション	目的

		`# Namespace imports`
		`#`
		`namespace import ::cisco::eem::*`
		`namespace import ::cisco::lib::*`
		`#`
		`package require xmlrpc`
		`puts "Did you get an error?"`

ステップ 6	**configure** 例： `RP/0/RSP0/CPU0:router# configure`	グローバル コンフィギュレーション モードを開始します。
ステップ 7	**event manager directory user library** *path* 例： `RP/0/RSP0/CPU0:router(config)# event manager directory user library disk2:/eem_library`	EEM ユーザ ライブラリのディレクトリを指定します。これは、ステップ 4, （51 ページ）のファイルがコピーされたディレクトリです。
ステップ 8	**event manager directory user policy** *path* 例： `RP/0/RSP0/CPU0:router(config)# event manager directory user policy disk2:/eem_policies`	EEM ユーザ ポリシーのディレクトリを指定します。これは、ステップ 5, （51 ページ）のファイルがコピーされたディレクトリです。
ステップ 9	**event manager policy** *policy-name* **username** *username* [**persist-time** [*seconds* | **infinite**] | **type** [**system** | **user**]] 例： `RP/0/RSP0/CPU0:router(config)# event manager policy packagetest.tcl username user_a`	ユーザ定義の EEM ポリシーを登録します。
ステップ 10	**event manager run** *policy* [*argument*] 例： `RP/0/RSP0/CPU0:router(config)# event manager run packagetest.tcl`	手動で EEM ポリシーを実行します。
ステップ 11	次のいずれかのコマンドを使用してください。 • **end** • **commit**	設定変更を保存します。 • **end** コマンドを実行すると、変更をコミットするように要求されます。 `Uncommitted changes found, commit them before exiting(yes/no/cancel)? [cancel]:`

	コマンドまたはアクション	目的

	例：
	`RP/0/RSP0/CPU0:router(config)# end`
	または
	`RP/0/RSP0/CPU0:router(config)# commit`
		◦ yes と入力すると、実行コンフィギュレーション ファイルに変更が保存され、コンフィギュレーション セッションが終了して、ルータが EXEC モードに戻ります。
		◦ no と入力すると、コンフィギュレーションセッションが終了して、ルータが EXEC モードに戻ります。変更はコミットされません。
		◦ cancel と入力すると、現在のコンフィギュレーションセッションが継続します。コンフィギュレーションセッションは終了せず、設定変更もコミットされません。
		• 実行コンフィギュレーション ファイルに設定変更を保存し、コンフィギュレーション セッションを継続するには、**commit** コマンドを使用します。

イベント管理ポリシーの設定例

ここでは、次の設定例を示します。

環境変数の設定：例

次の設定は、環境変数 cron_entry を設定します。

```
RP/0/RSP0/CPU0:router# configure
RP/0/RSP0/CPU0:router#(config)# event manager environment _cron_entry 0-59/2 0-23/1 * * 0-7
```

ユーザ定義 Embedded Event Manager ポリシーの登録：例

次の設定では、ユーザ定義イベント管理ポリシーを登録します。

```
RP/0/RSP0/CPU0:router# configure
RP/0/RSP0/CPU0:router(config)# event manager policy cron.tcl username tom user
```

使用可能なポリシーの表示：例

使用可能なポリシーを表示する **show event manager policy available** コマンドの出力例は、次のとおりです。

```
RP/0/RSP0/CPU0:router# show event manager policy available

No.  Type    Time Created                Name
1    system  Mon Mar 15 21:32:14 2004    periodic_diag_cmds.tcl
2    system  Mon Mar 15 21:32:14 2004    periodic_proc_avail.tcl
3    system  Mon Mar 15 21:32:16 2004    periodic_sh_log.tcl
4    system  Mon Mar 15 21:32:16 2004    tm_cli_cmd.tcl
5    system  Mon Mar 15 21:32:16 2004    tm_crash_hist.tcl
```

Embedded Event Manager プロセスの表示：例

信頼性メトリック データは、System Manager によって処理される各プロセスについて保持されます。 このデータには、プライマリまたはバックアップ ハードウェア カードで動作するスタンバイ プロセスが含まれています。 データは、ハードウェア カード ディスク ID、プロセス パス名、複数のインスタンスがあるプロセスの場合はプロセス インスタンスを組み合わせたものでインデックスが作成されたテーブルに記録されます。 信頼性メトリック データを表示する **show event manager metric process** コマンドの出力例は、次のとおりです。

```
RP/0/RSP0/CPU0:router# show event manager metric process all location 0/1/CPU0

====================================
job id: 78, node name: 0/1/CPU0
process name: wd-critical-mon, instance: 1
--------------------------------
last event type: process start
recent start time: Mon Sep 10 21:36:49 2007
recent normal end time: n/a
recent abnormal end time: n/a
number of times started: 1
number of times ended normally: 0
number of times ended abnormally: 0
most recent 10 process start times:
--------------------------
Mon Sep 10 21:36:49 2007
--------------------------

most recent 10 process end times and types:

cumulative process available time: 59 hours 33 minutes 42 seconds 638 milliseconds
cumulative process unavailable time: 0 hours 0 minutes 0 seconds 0 milliseconds
process availability:  1.000000000
number of abnormal ends within the past 60 minutes (since reload): 0
number of abnormal ends within the past 24 hours (since reload): 0
number of abnormal ends within the past 30 days (since reload): 0
====================================
job id: 56, node name: 0/1/CPU0
process name: dllmgr, instance: 1
--------------------------------
last event type: process start
recent start time: Mon Sep 10 21:36:49 2007
recent normal end time: n/a
recent abnormal end time: n/a
number of times started: 1
number of times ended normally: 0
number of times ended abnormally: 0
most recent 10 process start times:
```

```
--------------------------
Mon Sep 10 21:36:49 2007
--------------------------

most recent 10 process end times and types:

cumulative process available time: 59 hours 33 minutes 42 seconds 633 milliseconds
cumulative process unavailable time: 0 hours 0 minutes 0 seconds 0 milliseconds
process availability:  1.000000000
number of abnormal ends within the past 60 minutes (since reload): 0
number of abnormal ends within the past 24 hours (since reload): 0
number of abnormal ends within the past 30 days (since reload): 0
====================================
.
.
.
```

Tcl を使用した Embedded Event Manager（EEM）ポリシー記述の設定例

ここでは、次の設定例を示します。

EEM イベント ディテクタのデモ：例

この例では、サンプル ポリシーを使用して、Embedded Event Manager ポリシーの使用方法を示します。 後述のセクションで、サンプル ポリシーの使用方法について説明します。

EEM サンプル ポリシーの説明

設定例では、1 つのサンプル EEM ポリシーを取り上げています。 tm_cli_cmd.tcl は、設定可能な CRON エントリを使用して実行されます。 このポリシーでは、設定可能な CLI コマンドが実行され、結果が電子メールで送信されます。

サンプル ポリシーのイベント マネージャ環境変数

イベント マネージャ環境変数は、ポリシーの登録および実行の前に EEM ポリシーに対して外部定義された Tcl グローバル変数です。 サンプル ポリシーでは、3 つの電子メール環境変数が設定されている必要があります。_email_cc のみが省略可能です。 他の必須および任意の変数設定については、次の表で説明します。

次の表に、電子メール変数の一覧を示します。

表 *10*：サンプル ポリシーで使用される電子メール特有の環境変数

環境変数	説明	例
_domainname	デフォルト ドメイン名。	example.com

環境変数	説明	例
_email_server	電子メール送信に使用されるシンプル メール転送プロトコル（SMTP）メール サーバ。	mailserver.example.com
_email_to	電子メールの送信先アドレス。	engineering@example.com
_email_from	電子メールの送信元アドレス。	devtest@example.com
_email_cc	電子メールのコピーの送信先アドレス。	manager@example.com

次の表に、sl_intf_down.tcl サンプル ポリシーの実行前に設定する必要がある EEM 環境変数を示します。

表 *11*：*sl_intf_down.tcl* ポリシーで使用される環境変数

環境変数	説明	例
_config_cmd1	実行される最初のコンフィギュレーション コマンド。	**interface gigabitEthernet1/0/5/0**
_config_cmd2	実行される 2 番目のコンフィギュレーション コマンド。 この変数は任意で、指定する必要はありません。	**no shutdown**
_syslog_pattern	ポリシー実行時を決定するために syslog メッセージを比較するために使用する正規表現パターン マッチ文字列。	.*UPDOWN.*FastEthernet0/0.*

次の表に、tm_cli_cmd.tcl サンプル ポリシーの実行前に設定する必要がある EEM 環境変数を示します。

表 *12*：*tm_cli_cmd.tcl* ポリシーで使用される環境変数

環境変数	説明	例
_cron_entry	ポリシーが実行されるときを決定する CRON 仕様。	0-59/1 0-23/1 * * 0-7
_show_cmd	ポリシーの実行時に実行される CLI コマンド。	**show version**

次の表に、tm_crash_reporter.tcl サンプル ポリシーの実行前に設定する必要がある EEM 環境変数を示します。

表 *13*：*tm_crash_reporter.tcl* ポリシーで使用される環境変数

環境変数	説明	例
_crash_reporter_debug	tm_crash_reporter.tcl のデバッグ情報がイネーブルであるかどうかを決定する値。 この変数は任意で、指定する必要はありません。	1
_crash_reporter_url	クラッシュ レポートが送信される URL 位置。	http://www.example.com/fm/interface_tm.cgi

次の表に、tm_fsys_usage.tcl サンプル ポリシーの実行前に設定する必要がある EEM 環境変数を示します。

表 *14*：*tm_fsys_usage.tcl* ポリシーで使用される環境変数

環境変数	説明	例
_tm_fsys_usage_cron	event_register TCL コマンド拡張で使用される CRON 仕様。 指定されない場合、tm_fsys_usage.tcl ポリシーが 1 分に 1 回トリガーされます。 この変数は任意で、指定する必要はありません。	0-59/1 0-23/1 * * 0-7
_tm_fsys_usage_debug	この変数が値 1 に設定された場合、システムのすべてのエントリのディスク使用率情報が表示されます。 この変数は任意で、指定する必要はありません。	1
_tm_fsys_usage_freebytes	システムまたは特定のプレフィックスの空きバイト数しきい値。 空きスペースが所定の値を下回ると、警告が表示されます。 この変数は任意で、指定する必要はありません。	disk2:98000000

環境変数	説明	例
_tm_fsys_usage_percent	システムまたは特定のプレフィックスのディスク使用割合しきい値。 ディスク使用割合が所定の割合を超えると、警告が表示されます。 指定されない場合、すべてのシステムのデフォルトのディスク使用割合は、80% です。 この変数は任意で、指定する必要はありません。	nvram:25 disk2:5

一部の EEM ポリシーの登録

ポリシーの登録後に EEM 環境変数が変更された場合、一部の EEM ポリシーは、登録を解除し、再登録する必要があります。 ポリシーの開始時に表示される event_register_ *xxx* 文には、一部の EEM 環境変数が含まれ、この文は、ポリシーが実行される条件の確立に使用されます。 ポリシーの登録後に環境変数が変更された場合、条件は無効になります。 いかなるエラーも回避するには、ポリシーの登録を解除し、再登録する必要があります。 次の変数に影響が及ぼされます。

- _cron_entry in the tm_cli_cmd.tcl policy
- _syslog_pattern in the sl_intf_down.tcl policy

すべてのサンプル ポリシーの基本設定の詳細

Embedded Event Manager（EEM）から電子メールを送信できるようにするには、**hostname** コマンドと **ip domain-name** コマンドを設定する必要があります。 EEM 環境変数も設定する必要があります。 Cisco IOS XR ソフトウェア イメージのブート後、次の初期設定を使用し、ネットワークで適切な値を置き換えます。 tm_fsys_usage サンプル ポリシーの環境変数（表 14： tm_fsys_usage.tcl ポリシーで使用される環境変数, （57 ページ） を参照）はすべて任意で、ここではそのリストは示されていません。

```
hostname cpu
domain-name example.com
event manager environment _email_server ms.example.net
event manager environment _email_to username@example.net
event manager environment _email_from engineer@example.net
event manager environment _email_cc projectgroup@example.net
event manager environment _cron_entry 0-59/2 0-23/1 * * 0-7
event manager environment _show_cmd show event manager policy registered
event manager environment _syslog_pattern .*UPDOWN.*FastEthernet0/0
event manager environment _config_cmd1 interface Ethernet1/0
event manager environment _config_cmd2 no shutdown
event manager environment _crash_reporter_debug 1
event manager environment _crash_reporter_url
http://www.example.com/fm/interface_tm.cgi
end
```

サンプル ポリシーの使用

ここでは、4 つの Tcl サンプル ポリシーを使用する方法を示す次の設定シナリオについて説明します。

sl_intf_down.tcl サンプル ポリシーの実行

このサンプル ポリシーでは、特定のパターンで Syslog メッセージが記録されるときに設定を変更する機能について説明します。 ポリシーでは、イベントについての詳細情報が収集され、CLI ライブラリを使用して、EEM 環境変数 _config_cmd1 と、任意で _config_cmd2 で指定された、コンフィギュレーション コマンドが実行されます。 CLI コマンドの結果とともに、電子メール メッセージが送信されます。

次に、このポリシーの使用方法を示すサンプル設定について説明します。 EXEC モードで、**show event manager policy registered** コマンドを使用し、現在ポリシーが登録されていないことを確認します。 次のコマンドは **show event manager policy available** コマンドで、インストールできるポリシーが表示されます。 **configure** コマンドを入力してグローバル コンフィギュレーション モードを開始後に、**event manager policy** コマンドを使用して、EEM で sl_intf_down.tcl ポリシーを登録できます。 グローバル コンフィギュレーション モードを終了後、**show event manager policy registered** コマンドを再度入力し、ポリシーが登録されたことを確認します。

インターフェイスがダウンするときに、ポリシーが実行されます。 **show event manager environment** コマンドを入力し、現在の環境変数の値を表示します。 _syslog_pattern EEM 環境変数で指定されたインターフェイスのケーブルを取り外します（またはシャットダウンを設定します）。 インターフェイスがダウンし、インターフェイスがダウンしていることについての Syslog メッセージを記録する Syslog デーモンのプロンプトが表示されて、Syslog イベント ディテクタが呼び出されます。

Syslog イベント ディテクタによって、未解決のイベント仕様が見直され、インターフェイス ステータス変更に対する一致が検索されます。 EEM サーバに通知され、サーバでは、このイベント sl_intf_down.tcl を処理するために登録されたポリシーが実行されます。

```
enable
show event manager policy registered
show event manager policy available
configure terminal
 event manager policy sl_intf_down.tcl
 end
show event manager policy registered
show event manager environment
```

tm_cli_cmd.tcl サンプル ポリシーの実行

このサンプル ポリシーでは、定期的に CLI コマンドを実行し、結果を電子メールで送信する機能について説明します。 CRON 仕様「0-59/2 0-23/1 * * 0-7」を使用すると、このポリシーは、毎時 2 分目に実行されます。 ポリシーでは、イベントについての詳細情報が収集され、CLI ライブラリを使用して、EEM 環境変数 _show_cmd で指定された、コンフィギュレーション コマンドが実行されます。 CLI コマンドの結果とともに、電子メール メッセージが送信されます。

次に、このポリシーの使用方法を示すサンプル設定について説明します。 EXEC モードで、**show event manager policy registered** コマンドを入力し、現在ポリシーが登録されていないことを確認します。 次のコマンドは **show event manager policy available** コマンドで、インストールできるポリシーが表示されます。 **configure** コマンドを入力してグローバル コンフィギュレーション モードを開始後に、**event manager policy** コマンドを使用して、EEM で tm_cli_cmd.tcl ポリシーを登録できます。 グローバル コンフィギュレーション モードを終了後、**show event manager policy registered** コマンドを入力し、ポリシーが登録されたことを確認します。

EEM 環境変数 _cron_entry に設定されている CRON 文字列に従って、タイマー イベント ディテクタによって、定期的にこのケースのイベントがトリガーされます。 EEM サーバに通知され、サーバでは、このイベント tm_cli_cmd.tcl を処理するために登録されたポリシーが実行されます。

```
enable
show event manager policy registered
show event manager policy available
configure terminal
 event manager policy tm_cli_cmd.tcl
 end
show event manager policy registered
```

tm_crash_reporter.tcl サンプル ポリシーの実行

このサンプル ポリシーでは、ある URL へ HTTP 形式のクラッシュ レポートを送信する機能について説明します。 ポリシー登録がスタートアップ コンフィギュレーション ファイルに保存されている場合、ポリシーは、ブートの 5 秒後にトリガーされます。 トリガーされると、スクリプトによって、リロード原因の検索が試行されます。 リロードの原因がクラッシュの場合、ポリシーによって、関連する crashinfo ファイルが検索され、環境変数 _crash_reporter_url でユーザによって指定された URL へ、この情報が送信されます。 CGI スクリプト interface_tm.cgi は、tm_crash_reporter.tcl ポリシーから URL を受け取るために作成され、ターゲット URL マシン上のローカル データベースにクラッシュ情報が保存されます。

Perl CGI スクリプト interface_tm.cgi が作成され、HTTP サーバが含まれているマシン上で実行するために設計され、tm_crash_reporter.tcl ポリシーが実行されているルータからアクセスできます。 interface_tm.cgi スクリプトによって、tm_crash_reporter.tcl から渡されたデータが解析され、テキスト ファイルの末尾にクラッシュ情報が追加され、これによって、システムのすべてのクラッシュの履歴が作成されます。 さらに、各クラッシュの詳細情報は、ユーザが指定したクラッシュ データベース ディレクトリの 3 つのファイルに保存されます。 別の Perl CGI スクリプト crash_report_display.cgi は、interface_tm.cgi スクリプトによって作成されたデータベースに保存されている情報を表示するために作成されました。 crash_report_display.cgi スクリプトは、interface_tm.cgi が含まれているマシンと同じマシンに置く必要があります。 そのマシンでは、Internet Explorer または Netscape などのブラウザが実行されている必要があります。 crash_report_display.cgi スクリプトが実行されると、読み取り可能な形式でクラッシュ情報が表示されます。

次に、このポリシーの使用方法を示すサンプル設定について説明します。 EXEC モードで、**show event manager policy registered** コマンドを入力し、現在ポリシーが登録されていないことを確認します。 次に、**show event manager policy available** コマンドを入力し、インストールできるポリシーを表示します。 **configure** コマンドを入力してグローバル コンフィギュレーション モードを開始後に、**event manager policy** コマンドを使用して、EEM で tm_crash_reporter.tcl ポリシーを登

録できます。 グローバル コンフィギュレーション モードを終了後、**show event manager policy registered** コマンドを入力し、ポリシーが登録されたことを確認します。

```
enable
show event manager policy registered
show event manager policy available
configure terminal
 event manager policy tm_crash_reporter.tcl
 end
show event manager policy registered
```

tm_fsys_usage.tcl サンプル ポリシーの実行

このサンプル ポリシーでは、ディスク領域の使用状況を定期的にモニタし、値が設定可能なしきい値に近くなったときに Syslog を介してレポートする機能について説明します。

次に、このポリシーの使用方法を示すサンプル設定について説明します。 ユーザ EXEC モードで、**show event manager policy registered** コマンドを入力し、現在ポリシーが登録されていないことを確認します。 次に、**show event manager policy available** コマンドを入力し、インストールできるポリシーを表示します。 **configure** コマンドを入力してグローバル コンフィギュレーション モードを開始後に、**event manager policy** コマンドを使用して、EEM で tm_fsys_usage.tcl ポリシーを登録できます。 グローバル コンフィギュレーション モードを終了後、**show event manager policy registered** コマンドを再度入力し、ポリシーが登録されたことを確認します。 tm_fsys_usage.tcl ポリシーで使用される任意の環境変数のいずれかを設定した場合、**show event manager environment** コマンドによって、設定された変数が表示されます。

```
enable
show event manager policy registered
show event manager policy available
configure terminal
 event manager policy tm_fsys_usage.tcl
 end
show event manager policy registered
show event manager environment
```

Tcl でポリシーをプログラミングするサンプル スクリプト例

ここでは、EEM システム ポリシーとして含まれている 2 つのサンプル ポリシーについて説明します。 これらのポリシーの詳細については、EEM イベント ディテクタのデモ：例，（55 ページ）を参照してください。

tm_cli_cmd.tcl サンプル ポリシー

次に、設定可能なCRONエントリが実行されるサンプルポリシーについて説明します。 ポリシーでは、設定可能な Cisco IOS XR ソフトウェア CLI コマンドが実行され、結果が電子メールで送信されます。 タイムスタンプとともに出力が末尾に追加される任意のログファイルを定義することができます。

```
::cisco::eem::event_register_timer cron name crontimer2 cron_entry $_cron_entry maxrun 240
#-------------------------------------------------------------------
# EEM policy that will periodically execute a cli command and email the
# results to a user.
```

```
#
# July 2005, Cisco EEM team
#
# Copyright (c) 2005 by cisco Systems, Inc.
# All rights reserved.
#------------------------------------------------------------------
### The following EEM environment variables are used:
###
### _cron_entry (mandatory)            - A CRON specification that determines
###                                      when the policy will run. See the
###                                      IOS XR Embedded Event Manager
###                                      documentation for more information
###                                      on how to specify a cron entry.
### Example: _cron_entry                 0-59/1 0-23/1 * * 0-7
###
### _log_file (mandatory without _email_....)
###                                    - A filename to append the output to.
###                                      If this variable is defined, the
###                                      output is appended to the specified
###                                      file with a timestamp added.
### Example: _log_file                   disk0:/my_file.log
###
### _email_server (mandatory without _log_file)
###                                    - A Simple Mail Transfer Protocol (SMTP)
###                                      mail server used to send e-mail.
### Example: _email_server               mailserver.example.com
###
### _email_from (mandatory without _log_file)
###                                    - The address from which e-mail is sent.
### Example: _email_from                 devtest@example.com
###
### _email_to (mandatory without _log_file)
###                                    - The address to which e-mail is sent.
### Example: _email_to                   engineering@example.com
###
### _email_cc (optional)               - The address to which the e-mail must
###                                      be copied.
### Example: _email_cc                   manager@example.com
###
### _show_cmd (mandatory)              - The CLI command to be executed when
###                                      the policy is run.
### Example: _show_cmd                   show version
###
# check if all required environment variables exist
# If any required environment variable does not exist, print out an error msg and quit
if {![info exists _log_file]} {
    if {![info exists _email_server]} {
set result \
"Policy cannot be run: variable _log_file or _email_server has not been set"
error $result $errorInfo
    }
    if {![info exists _email_from]} {
set result \
"Policy cannot be run: variable _log_file or _email_from has not been set"
error $result $errorInfo
    }
    if {![info exists _email_to]} {
set result \
"Policy cannot be run: variable _log_file ore _email_to has not been set"
error $result $errorInfo
    }
    if {![info exists _email_cc]} {
#_email_cc is an option, must set to empty string if not set.
set _email_cc ""
    }
}
if {![info exists _show_cmd]} {
    set result \
        "Policy cannot be run: variable _show_cmd has not been set"
    error $result $errorInfo
}
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
```

```
# query the event info and log a message
array set arr_einfo [event_reqinfo]
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
global timer_type timer_time_sec
set timer_type $arr_einfo(timer_type)
set timer_time_sec $arr_einfo(timer_time_sec)
# log a message
set msg [format "timer event: timer type %s, time expired %s" \
        $timer_type [clock format $timer_time_sec]]
action_syslog priority info msg $msg
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
$_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
# 1. execute the command
if [catch {cli_open} result] {
    error $result $errorInfo
} else {
    array set cli1 $result
}

# save exact execution time for command
set time_now [clock seconds]
# execute command
if [catch {cli_exec $cli1(fd) $_show_cmd} result] {
    error $result $errorInfo
} else {
    set cmd_output $result
    # format output: remove trailing router prompt
    regexp {\n*(.*\n)([^\n]*)$} $result dummy cmd_output
}
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
    error $result $errorInfo
}
# 2. log the success of the CLI command
set msg [format "Command \"%s\" executed successfully" $_show_cmd]
action_syslog priority info msg $msg
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
# 3. if _log_file is defined, then attach it to the file
if {[info exists _log_file]} {
    # attach output to file
    if [catch {open $_log_file a+} result] {
        error $result
    }
    set fileD $result
    # save timestamp of command execution
    #      (Format = 00:53:44 PDT Mon May 02 2005)
    set time_now [clock format $time_now -format "%T %Z %a %b %d %Y"]
    puts $fileD "%%% Timestamp = $time_now"
    puts $fileD $cmd_output
    close $fileD
}
# 4. if _email_server is defined send the email out
if {[info exists _email_server]} {
    set routername [info hostname]
    if {[string match "" $routername]} {
error "Host name is not configured"
    }
    if [catch {smtp_subst [file join $tcl_library email_template_cmd.tm]} \
    result] {
error $result $errorInfo
    }
    if [catch {smtp_send_email $result} result] {
error $result $errorInfo
```

```
    }
}
```

sl_intf_down.tcl サンプル ポリシー

次に、設定可能な Syslog メッセージが記録されるときに実行されるサンプル ポリシーを示します。 ポリシーでは、設定可能な CLI コマンドが実行され、結果が電子メールで送信されます。

```
::cisco::eem::event_register_syslog occurs 1 pattern $_syslog_pattern maxrun 90
#------------------------------------------------------------------
# EEM policy to monitor for a specified syslog message.
# Designed to be used for syslog interface-down messages.
# When event is triggered, the given config commands will be run.
#
# July 2005, Cisco EEM team
#
# Copyright (c) 2005 by cisco Systems, Inc.
# All rights reserved.
#------------------------------------------------------------------
### The following EEM environment variables are used:
###
### _syslog_pattern (mandatory)        - A regular expression pattern match string
###                                      that is used to compare syslog messages
###                                      to determine when policy runs
### Example: _syslog_pattern             .*UPDOWN.*FastEthernet0/0.*
###
### _email_server (mandatory)          - A Simple Mail Transfer Protocol (SMTP)
###                                      mail server used to send e-mail.
### Example: _email_server               mailserver.example.com
###
### _email_from (mandatory)            - The address from which e-mail is sent.
### Example: _email_from                 devtest@example.com
###
### _email_to (mandatory)              - The address to which e-mail is sent.
### Example: _email_to                   engineering@example.com
###
### _email_cc (optional)               - The address to which the e-mail must
###                                      be copied.
### Example: _email_cc                   manager@example.com
###
### _config_cmd1 (optional)            - The first configuration command that
###                                      is executed.
### Example: _config_cmd1                interface Ethernet1/0
###
### _config_cmd2 (optional)            - The second configuration command that
###                                      is executed.
### Example: _config_cmd2                no shutdown
###
# check if all the env variables we need exist
# If any of them doesn't exist, print out an error msg and quit
if {![info exists _email_server]} {
    set result \
        "Policy cannot be run: variable _email_server has not been set"
    error $result $errorInfo
}
if {![info exists _email_from]} {
    set result \
        "Policy cannot be run: variable _email_from has not been set"
    error $result $errorInfo
}
if {![info exists _email_to]} {
    set result \
        "Policy cannot be run: variable _email_to has not been set"
    error $result $errorInfo
}
if {![info exists _email_cc]} {
     #_email_cc is an option, must set to empty string if not set.
     set _email_cc ""
}
```

```
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
# 1. query the information of latest triggered eem event
array set arr_einfo [event_reqinfo]
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
      $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
set msg $arr_einfo(msg)
set config_cmds ""
# 2. execute the user-defined config commands
if [catch {cli_open} result] {
    error $result $errorInfo
} else {
    array set cli1 $result
}

if [catch {cli_exec $cli1(fd) "config t"} result] {
    error $result $errorInfo
}
if {[info exists _config_cmd1]} {
    if [catch {cli_exec $cli1(fd) $_config_cmd1} result] {
        error $result $errorInfo
    }
    append config_cmds $_config_cmd1
}
if {[info exists _config_cmd2]} {
    if [catch {cli_exec $cli1(fd) $_config_cmd2} result] {
        error $result $errorInfo
    }
    append config_cmds "\n"
    append config_cmds $_config_cmd2
}
if [catch {cli_exec $cli1(fd) "end"} result] {
    error $result $errorInfo
}
if [catch {cli_close $cli1(fd) $cli1(tty_id)} result] {
    error $result $errorInfo
}
after 60000
# 3. send the notification email
set routername [info hostname]
if {[string match "" $routername]} {
    error "Host name is not configured"
}
if [catch {smtp_subst [file join $tcl_library email_template_cfg.tm]} result] {
    error $result $errorInfo
}
if [catch {smtp_send_email $result} result] {
    error $result $errorInfo
}
```

次に、前述の EEM サンプル ポリシーで使用される電子メール テンプレート ファイルの使用例を示します。

```
email_template_cfg.tm
Mailservername: $_email_server
From: $_email_from
To: $_email_to
Cc: $_email_cc
Subject: From router $routername: Periodic $_show_cmd Output
$cmd_output
```

Tcl set コマンド操作のトレース：例

Tcl は、融通性のある言語です。 Tcl の融通性の 1 つは、コマンドを上書きできることです。 この例では、Tcl **set** コマンドの名前が _set に変更されます。また、テキスト「setting」が含まれるメッセージを表示し、設定しているスカラ変数を末尾に追加する、新バージョンの **set** コマンドが作成されます。 この例を使用すると、設定しているスカラ変数のすべてのインスタンスをトレースできます。

```
rename set _set
proc set {var args} {
   puts [list setting $var $args]
   uplevel _set $var $args
};
When this is placed in a policy, a message is displayed anytime a scalar variable is set,
for example:

02:17:58: sl_intf_down.tcl[0]: setting test_var 1
```

その他の参考資料

次の項では、Embedded Event Manager ポリシーの設定および管理についての関連資料を示します。

関連資料

関連項目	参照先
Embedded Event Manager コマンド	*Cisco ASR 9000 Series Aggregation Services Router System Monitoring Command Reference* の「*Embedded Event Manager Commands*」モジュール
ルート プロセッサ フェールオーバー コマンド	*Cisco ASR 9000 Series Aggregation Services Router Interface and Hardware Component Command Reference* の「Hardware Redundancy and Node Administration Commands」モジュール
Cisco IOS XR XML API 参考資料	*Cisco IOS XR XML API Guide*
Cisco IOS XR スタートアップ参考資料	*Cisco ASR 9000 Series Aggregation Services Router Getting Started Guide*
ユーザ グループとタスク ID に関する情報	*Cisco ASR 9000 Series Aggregation Services Router System Security Configuration Guide* の「*Configuring AAA Services*」モジュール

標準

標準	タイトル
この機能でサポートされる新規の標準または変更された標準はありません。また、既存の標準のサポートは変更されていません。	—

MIB

MIB	MIB のリンク
—	Cisco IOS XR ソフトウェアを使用している MIB を特定してダウンロードするには、次の URL にある Cisco MIB Locator を使用し、[Cisco Access Products] メニューからプラットフォームを選択します。http://cisco.com/public/sw-center/netmgmt/cmtk/mibs.shtml

RFC

RFC	タイトル
この機能によりサポートされた新規 RFC または改訂 RFC はありません。また、この機能による既存RFCのサポートに変更はありません。	—

シスコのテクニカル サポート

説明	リンク
シスコのテクニカルサポートWebサイトには、数千ページに及ぶ検索可能な技術情報があります。製品、テクノロジー、ソリューション、技術的なヒント、およびツールへのリンクもあります。 Cisco.com に登録済みのユーザは、このページから詳細情報にアクセスできます。	http://www.cisco.com/en/US/support/index.html

Embedded Event Manager ポリシー Tcl コマンド拡張リファレンス

ここでは、次の EEM ポリシーの Tcl コマンド拡張カテゴリについて説明します。

（注） すべての EEM Tcl コマンド拡張について、エラーがあった場合、戻される Tcl 結果文字列には、エラー情報が含まれます。

（注） 数値範囲が指定されていない引数は、–2147483648 から 2147483647 までの整数から取得されます。

次の表記法が、Tcl コマンド拡張ページで説明されている構文に使用されます。

- 任意の引数は、たとえば次の例のように、角カッコ内に示されます。

```
[type ?]
```

- 疑問符（?）は 入力する変数を表します。
- 引数間の選択肢は、たとえば次の例のように、パイプ文字で示されます。

```
[queue_priority low|normal|high]
```

Embedded Event Manager イベント登録 Tcl コマンド拡張

次の EEM イベント登録 Tcl コマンド拡張がサポートされています。

event_register_appl

アプリケーション イベントの登録を行います。 この Tcl コマンド拡張を使用すると、アプリケーション イベントがトリガーされ、続いて event_publish Tcl コマンド拡張の別のポリシーが実行されるときに、ポリシーが実行されます。event_publish コマンド拡張によって、アプリケーション イベントがパブリッシュされます。

アプリケーション イベントを登録するには、サブシステムを指定する必要があります。 Tcl ポリシーまたは内部 EEM API のいずれかによって、アプリケーション イベントをパブリッシュできます。イベントがポリシーによってパブリッシュされている場合、ポリシーで予約される *sub_system* 引数は 798 です。

構文

```
event_register_appl [sub_system ?] [type ?] [queue_priority low|normal|high] [maxrun ?]
[nice 0|1]
```

引数

sub_system	（任意）アプリケーションイベントをパブリッシュしたEEMポリシーに割り当てられる番号。他のすべての番号はCiscoで使用するために予約されており、番号は798に設定されます。この引数が指定されない場合、すべてのコンポーネントが照会されます。
type	（任意）指定されたイベント内のイベントサブタイプ。*sub_system*引数および*type*引数によって、アプリケーションイベントが一意に識別されます。この引数が指定されない場合、すべてのタイプが照会されます。この引数を指定する場合、1～4294967295の整数を選択する必要があります。 パブリッシュと登録を動作させるには、**event_publish**コマンド拡張と**event_register_appl**コマンド拡張との間のコンポーネントとタイプが一致する必要があります。
queue_priority	（任意）スクリプトのキューイングに使用されるプライオリティ レベル。normalは、lowプライオリティよりも高く、highプライオリティよりも低いプライオリティです。このプライオリティは実行プライオリティではなく、キューイング プライオリティです。 この引数が指定されない場合、デフォルトのプライオリティはnormal です。
maxrun	（任意）スクリプトの最大ランタイム（SSSSSSSSSS[.MMM] 形式で指定します。SSSSSSSSSSは、0～4294967295の秒数を表す整数で、MMMは0～999のミリ秒数を表す整数である必要があります）。この引数が指定されない場合、デフォルトの20秒ランタイム制限が使用されます。

nice	（任意）ポリシー実行時間のプライオリティ設定。 *nice* 引数が 1 に設定されている場合、ポリシーは、デフォルトプライオリティよりも低い実行時プライオリティで実行されます。デフォルト値は 0 です。

複数の条件が存在する場合、すべての条件が満たされたときに、アプリケーション イベントが発生します。

結果文字列

なし

_cerrno を設定

No

event_register_counter

パブリッシャとサブスクライバの両方として、カウンタ イベントの登録を行います。 この Tcl コマンド拡張を使用すると、しきい値に近くなった名前付きカウンタに基づいて、ポリシーが実行されます。サブスクライバとして、このイベントカウンタによって、登録に必要なカウンタの名前が指定され、別のポリシーまたは別のプロセスに依存して、カウンタが実際に操作されます。たとえば、policyB がカウンタ ポリシーとして動作し、policyA（カウンタ ポリシーは不要ですが）では、register_counter、counter_modify、または unregister_counter の各 Tcl コマンド拡張を使用して、policyB で定義されているカウンタが操作されます。

構文

```
event_register_counter name ? entry_op gt|ge|eq|ne|lt|le entry_val ?
exit_op gt|ge|eq|ne|lt|le exit_val ? [queue_priority low|normal|high]
[maxrun ?] [nice 0|1]
```

引数

name	（必須）カウンタの名前。
entry_op	（必須）現在のカウンタの値を開始値と比較するために使用される開始比較演算子。真の場合、イベントが発生し、終了基準を満たすまでイベントモニタリングがディセーブルにされます。
entry_val	（必須）カウンタイベントを発生させる必要があるかどうかを判断するために、現在のカウンタの値を比較する必要がある値。

exit_op	（必須）現在のカウンタの値を終了値と比較するために使用される終了比較演算子。真の場合、このイベントのイベントモニタリングが再度イネーブルにされます。
exit_val	（必須）終了基準を満たすかどうかを判断するために、現在のカウンタの値を比較する必要がある値。
queue_priority	（任意）スクリプトのキューイングに使用されるプライオリティ レベル。normal は、low プライオリティよりも高く、high プライオリティよりも低いプライオリティです。 このプライオリティは実行プライオリティではなく、キューイング プライオリティです。 この引数が指定されない場合、デフォルトのプライオリティは normal です。
maxrun	（任意）スクリプトの最大ランタイム（SSSSSSSSSS[.MMM] 形式で指定します。SSSSSSSSSS は、0 ～ 4294967295 の秒数を表す整数で、MMM は 0 ～ 999 のミリ秒数を表す整数である必要があります）。 この引数が指定されない場合、デフォルトの 20 秒ランタイム制限が使用されます。
nice	（任意）ポリシー実行時間のプライオリティ設定。 *nice* 引数が 1 に設定されている場合、ポリシーは、デフォルトプライオリティよりも低い実行時プライオリティで実行されます。デフォルト値は 0 です。

結果文字列

なし

_cerrno を設定

No

event_register_hardware

ハードウェア イベントと条件で指定される環境モニタリング ハードウェア デバイス用に登録します。

構文

```
event_register_hardware env_device ? env_cond ?
[priority normal|low|high] [maxrun_sec ?] [maxrun_nsec ?] [nice 0|1]
```

引数

env_device	（必須）モニタリング用に使用する環境デバイス。1 ～ 2147483647 の範囲の整数であることが必要です。これは、複数のタイプの環境デバイスをモニタするビットマスクです。 サポートされるデバイスとその対応するビットマスクの一覧を次に示します。 • 0x0001 シャーシ • 0x0002 バックプレーン • 0x0004 スロット • 0x0008 カード • 0x0010 ポート • 0x0020 ファン • 0x0040 電源のグループ • 0x0080 電源 • 0x0100 センサー 複数のデバイスをモニタするには、ビット単位の OR を行います。

env_cond	（必須）モニタする環境条件。これは、複数のタイプの環境条件をモニタするビットマスクです。 サポートされる環境条件とその対応するビットマスクの一覧を次に示します。 • 0x0001 低警告 • 0x0002 高警告 • 0x0004 警告 • 0x0010 低クリティカル • 0x0020 高クリティカル • 0x0040 クリティカル • 0x0100 プレシャットダウン • 0x0200 シャットダウン
priority	（任意）スクリプトをキューに格納する優先順位レベル。指定しない場合、デフォルトでは通常の優先順位が使用されます。
maxrun_sec、maxrun_nsec	（任意）秒またはナノ秒単位で指定される最大実行時間。 0 ～ 2147483647 の範囲の整数であることが必要です。指定しない場合、デフォルトの 20 秒の実行時間制限が使用されます。
nice	（任意）秒またはナノ秒単位で指定される最大実行時間。 0 ～ 2147483647 の範囲の整数であることが必要です。指定しない場合、デフォルトの 20 秒の実行時間制限が使用されます。

結果文字列

なし

_cerrno を設定

No

event_register_none

event manager run コマンドによってトリガーされるイベントの登録を行います。 これらのイベントは、このイベントをスクリーニングする None イベント ディテクタによって処理されます。

構文

```
event_register_none [queue_priority low|normal|high] [maxrun ?] [nice 0|1]
```

引数

queue_priority	（任意）スクリプトのキューイングに使用されるプライオリティ レベル。normal は、low プライオリティよりも高く、high プライオリティよりも低いプライオリティです。このプライオリティは実行プライオリティではなく、キューイング プライオリティです。 この引数が指定されない場合、デフォルトのプライオリティは normal です。
maxrun	（任意）スクリプトの最大ランタイム（SSSSSSSSSS[.MMM] 形式で指定します。SSSSSSSSSS は、0 ～ 4294967295 の秒数を表す整数で、MMM は 0 ～ 999 のミリ秒数を表す整数である必要があります）。この引数が指定されない場合、デフォルトの 20 秒ランタイム制限が使用されます。
nice	（任意）ポリシー実行時間のプライオリティ設定。 *nice* 引数が 1 に設定されている場合、ポリシーは、デフォルトプライオリティよりも低い実行時プライオリティで実行されます。デフォルト値は 0 です。

結果文字列

なし

_cerrno を設定

No

event_register_oir

活性挿抜（OIR）イベントの登録を行います。 この Tcl コマンド拡張を使用すると、ハードウェア カード OIR イベントの発生時に発生するイベントに基づいて、ポリシーが実行されます。 これらのイベントは、このイベントをスクリーニングする OIR イベント ディテクタによって処理されます。

構文

```
event_register_oir [queue_priority low|normal|high] [maxrun ?] [nice 0|1]
```

引数

queue_priority	（任意）スクリプトのキューイングに使用されるプライオリティ レベル。normal は、low プライオリティよりも高く、high プライオリティよりも低いプライオリティです。 このプライオリティは実行プライオリティではなく、キューイング プライオリティです。 この引数が指定されない場合、デフォルトのプライオリティは normal です。
maxrun	（任意）スクリプトの最大ランタイム（SSSSSSSSSS[.MMM] 形式で指定します。SSSSSSSSSS は、0 ～ 4294967295 の秒数を表す整数で、MMM は 0 ～ 999 のミリ秒数を表す整数である必要があります）。 この引数が指定されない場合、デフォルトの 20 秒ランタイム制限が使用されます。
nice	（任意）ポリシー実行時間のプライオリティ設定。 *nice* 引数が 1 に設定されている場合、ポリシーは、デフォルトプライオリティよりも低い実行時プライオリティで実行されます。デフォルト値は 0 です。

結果文字列

なし

_cerrno を設定

No

event_register_process

プロセス イベントの登録を行います。 この Tcl コマンド拡張を使用すると、Cisco IOS XR ソフトウェア モジュール方式プロセスの開始時と停止時に発生するイベントに基づいて、ポリシーが実行されます。 これらのイベントは、このイベントをスクリーニングする System Manager イベント ディテクタによって処理されます。 この Tcl コマンド拡張は、ソフトウェア モジュール方式イメージでのみサポートされます。

構文

```
event_register_process abort|term|start
[job_id ?] [instance ?] [path ?] [node ?]
[queue_priority low|normal|high] [maxrun ?] [nice 0|1] [tag?]
```

引数

abort	（必須）プロセスの異常な終了。ゼロではない終了ステータスでの終了、カーネル生成信号の受信、またはユーザ要求のために送信されない SIGTERM 信号または SIGKILL 信号の受信のため、プロセスが強制終了されることがあります。
term	（必須）プロセスの正常な終了。
start	（必須）プロセスの開始。
job_id	（任意）プロセスイベントをパブリッシュした EEM ポリシーに割り当てられる番号。 他のすべての番号は Cisco での使用のために予約されているため、番号は 798 に設定されます。
instance	（任意）プロセス インスタンス ID。 指定される場合、この引数は、1 ～ 4294967295 の範囲の整数である必要があります。
path	（任意）プロセスパス名（正規表現文字列）。
node	（任意）ノード名は、「node」という語句と、それに続く、次の形式を使用してスラッシュ（/）で区切られた 2 つのフィールドで構成される、文字列です。 node<slot-number>/<cpu-number> slot-number は、ハードウェア スロット番号です。 cpu-number は、ハードウェア CPU 番号です。 たとえば、スロット 0 にある Cisco Catalyst 6500 シリーズ スイッチのスーパーバイザ カードの SP CPU は、node0/0 と指定されます。 たとえば、スロット 0 にある Cisco Catalyst 6500 シリーズ スイッチのスーパーバイザ カードの RP CPU は、node0/1 と指定されます。 *node* 引数が指定されない場合、デフォルトのノード指定は、常に、すべての該当するノードを表す正規表現パターン マッチ * です。

queue_priority	（任意）スクリプトのキューイングに使用されるプライオリティ レベル。normal は、low プライオリティよりも高く、high プライオリティよりも低いプライオリティです。 このプライオリティは実行プライオリティではなく、キューイング プライオリティです。 この引数が指定されない場合、デフォルトのプライオリティは normal です。
maxrun	（任意）スクリプトの最大ランタイム（SSSSSSSSSS[.MMM] 形式で指定します。SSSSSSSSSS は、0 ～ 4294967295 の秒数を表す整数で、MMM は 0 ～ 999 のミリ秒数を表す整数である必要があります）。 この引数が指定されない場合、デフォルトの 20 秒ランタイム制限が使用されます。
nice	（任意）ポリシー実行時間のプライオリティ設定。 *nice* 引数が 1 に設定されている場合、ポリシーは、デフォルトプライオリティよりも低い実行時プライオリティで実行されます。デフォルト値は 0 です。
tag	タグは指定できますが無視されます。 タグ オプションを指定した Cisco IOS EEM スクリプトは、エラーになることなく Cisco IOS XR ソフトウェア環境で実行できます。 Cisco IOS XR ソフトウェアは複数のイベントをサポートしていないため、タグの効果はありません。

任意の引数が指定されない場合、イベントは、引数のすべての可能な値に対して照会されます。複数の引数が存在する場合、すべての条件が一致したときに、プロセス イベントが発生します。

結果文字列

なし

_cerrno を設定

No

event_register_stat

統計情報イベントの登録を行います。 この Tcl コマンド拡張を使用すると、特定の統計カウンタが定義されたしきい値を超えた場合にポリシーを実行できます。

EEM キーワードが監視する統計カウンタを一意に識別するために、次の 3 つのフィールドがあります。

- 引数名に対応するデータ要素名。 たとえば、ifstats-generic 名がインターフェイス汎用統計として定義されています。
- データ要素の最初の修飾子は、*modifier_1* 引数に対応します。 たとえば、Ethernet1_0 は ifstats-generic の最初の修飾子として定義されており、インターフェイス汎用統計情報がイーサネット インターフェイス固有であることを修飾します。
- データ要素の 2 番目の修飾子は、*modifier_2* 引数に対応します。 たとえば、input-ptks は、ifstats-generic の 2 番目の修飾子として定義されており、特定のイーサネット インターフェイスのインターフェイス統計情報を、受信パケットの数としてさらに修飾します。

構文

```
event_register_stat name ? [modifier_1 ?] [modifier_2 ?]

entry_op gt|ge|eq|ne|lt|le entry_val ? [exit_comb or|and]

[exit_op gt|ge|eq|ne|lt|le] [exit_val ?] [exit_time_sec ?] [exit_time_nsec ?]
[poll_interval_sec ?] [poll_interval_nsec ?] [priority normal|low|high]
[maxrun_sec ?] [maxrun_nsec ?] [nice 0|1] [tag ?]
```

引数

name	（必須）統計データ要素名。
modifier_1	インターフェイス統計情報では必須ですが、それ以外ではオプションです。インターフェイス統計情報の場合、この変数はインターフェイス名です。インターフェイス名を取得するには、**show interface brief** コマンドを使用します。 このコマンドは、スラッシュ（/）で指定された、現在設定されているすべてのインターフェイス名の一覧を表示します（たとえば Ethernet 1/0）。 このインターフェイスを *modifier_1* 引数に対して設定するには、スラッシュをアンダースコアに変更します。

modifier_2	インターフェイス統計情報では必須ですが、それ以外ではオプションです。インターフェイス統計情報の場合、この変数はインターフェイス統計情報名です。インターフェイス統計情報名を取得するには、**show event manager statistics -table** コマンドを **all** キーワードとともに使用して、すべての統計情報のクラスを表示します。その後、**show event manager statistics -table** コマンドを *name* 引数とともに使用して、modifier_2 の特定の統計情報名を取得します。
entry_op	（必須）現在の統計値をエントリ値と比較するために使用するエントリ比較演算子。 true の場合イベントが発生し、終了条件が満たされるまでイベント監視が無効になります。
entry_val	（必須）統計情報イベントを発生させるかどうかを判断するために、現在の統計情報カウンタの値を比較する値。
exit_comb	（必須）イベント監視を再度イネーブルにできるように、終了条件が満たされているかどうかを判断するために必要な、終了条件テストの組み合わせを示す終了組み合わせ演算子。 条件が満たされている場合、終了条件を満たすためには、終了値テストと終了時間テストに合格する必要があります。または、終了値テストまたは終了時間テストのいずれかが終了条件を満たします。 *exit_comb* および *exit_op*、*exit_val* 引数（*exit_time_sec* 引数または *exit_time_nsec* 引数）が存在する必要があります。 *exit_comb* 引数または（*exit_op* および *exit_val* 引数）または（*exit_time_sec* 引数または *exit_time_nsec* 引数）が存在する必要があります。
exit_op	現在の統計値を終了値と比較するために使用する終了比較演算子。 true の場合、このイベントのイベント監視が再度イネーブルになります。
exit_val	終了条件が満たされているかどうかを判断するために、現在の統計情報カウンタの値を比較する値。

exit_time_sec exit_time_nse	イベント監視を再度イネーブルにする場合に、イベントが発生してからの POSIX タイマー ユニットの数。 整数は、0 ～ 2147483647 の範囲にしてください。
poll_interval_sec poll_interval_nsec	引数 *poll_interval_sec* と *poll_interval_nsec* のいずれかを指定する必要があります。 間隔は、POSIX 時間単位での連続するポーリングの間にある必要があります。 現在 1 秒以上に制限されています。 整数は、0 ～ 2147483647 の範囲にしてください。
priority	（任意）スクリプトに対してキューに格納される優先度。 指定しない場合、デフォルトでは通常の優先順位が使用されます。
maxrun_sec、 maxrun_nsec	（任意）秒またはナノ秒単位で指定される最大実行時間。 指定しない場合、デフォルトとして 20 秒のランタイム制限が使用されます。 整数は、0 ～ 2147483647 の範囲にしてください。
nice	（任意）*nice* 引数が値 1 に設定されている場合、ポリシーは、デフォルト プライオリティよりも低い実行時プライオリティで実行されます。 デフォルト値は 0 です。
tag	タグは指定できますが無視されます。タグ オプションを指定した Cisco IOS EEM スクリプトは、エラーになることなく Cisco IOS XR ソフトウェア環境で実行できます。 Cisco IOS XR ソフトウェアは複数のイベントをサポートしていないため、タグの効果はありません。

（注） 終了条件には、時間ベース、値ベース、または両方を指定できます。 終了条件が満たされるまでイベント監視は再度イネーブルになりません。

複数の条件が存在する場合、すべての条件が満たされたときに、統計情報イベントが発生します。

結果文字列

なし

_cerrno を設定

No

event_register_syslog

Syslog イベントの登録を行います。 この Tcl コマンド拡張を使用すると、一定の時間内に一定回数の発生後、特定パターンの Syslog メッセージが記録されるときに、ポリシーがトリガーされます。

構文

```
event_register_syslog [occurs ?] [period ?] pattern ?
[priority all|emergencies|alerts|critical|errors|warnings|notifications|
informational|debugging|0|1|2|3|4|5|6|7]
[queue_priority low|normal|high]
[severity_fatal] [severity_critical] [severity_major]
[severity_minor] [severity_warning] [severity_notification]
[severity_normal] [severity_debugging]
[maxrun ?] [nice 0|1]
```

引数

occurs	（任意）イベントが発生する前の発生回数。この引数が指定されない場合、イベントは 1 回目から発生します。 指定される場合、0 より大きい値を指定する必要があります。
period	（任意）イベントを発生させるために取る必要がある 1 つまたは複数のイベントの間の、秒単位およびミリ秒単位の時間の間隔（SSSSSSSSSS[.MMM] 形式で指定します。SSSSSSSSSS は、0 ～ 4294967295 の秒数を表す整数で、MMM は 0 ～ 999 のミリ秒数を表す整数である必要があります）。 この引数が指定されない場合、期間チェックは適用されません。
pattern	（必須）Syslog メッセージ パターン マッチの実行に使用される正規表現。 この引数は、記録された Syslog メッセージを指定するためにポリシーによって使用されます。
priority	（任意）スクリーニングされるメッセージのプライオリティ。 この引数が指定される場合、指定されたロギング プライオリティ レベルまたはそれ以下メッセージのみがスクリーニングされます。 この引数が指定されない場合、デフォルトのプライオリティは 0 です。

queue_priority	（任意）スクリプトのキューイングに使用されるプライオリティ レベル。normal は、low プライオリティよりも高く、high プライオリティよりも低いプライオリティです。このプライオリティは実行プライオリティではなく、キューイング プライオリティです。 この引数が指定されない場合、デフォルトのプライオリティは normal です。
maxrun	（任意）スクリプトの最大ランタイム（SSSSSSSSSS[.MMM] 形式で指定します。SSSSSSSSSS は、0 ～ 4294967295 の秒数を表す整数で、MMM は 0 ～ 999 のミリ秒数を表す整数である必要があります）。 この引数が指定されない場合、デフォルトの 20 秒ランタイム制限が使用されます。
nice	（任意）ポリシー実行時間のプライオリティ設定。 *nice* 引数が 1 に設定されている場合、ポリシーは、デフォルトプライオリティよりも低い実行時プライオリティで実行されます。デフォルト値は 0 です。

複数の条件が存在する場合、すべての条件が一致したときに、Syslog イベントが発生します。

表 *15*：*Syslog* イベントの重大度のマッピング

重大度のキーワード	Syslog のプライオリティ	説明
severity_fatal	LOG_EMERG (0)	システムが使用不可能な状態。
severity_critical	LOG_ALERT (1)	クリティカル条件で、即時対応が必要であることを示す
severity_major	LOG_CRIT (2)	重大な状態。
severity_minor	LOG_ERR (3)	軽微な状態。
severity_warning	LOG_WARNING (4)	警告状態。
severity_notification	LOG_NOTICE (5)	基本的な通知、情報メッセージ
severity_normal	LOG_INFO (6)	正常なイベント、正常な状態に戻ったことを伝える

重大度のキーワード	Syslog のプライオリティ	説明
severity_debugging	LOG_DEBUG (7)	デバッグ メッセージ。

結果文字列

なし

_cerrno を設定

No

event_register_timer

パブリッシャとサブスクライバの両方として、タイマーを作成し、タイマー イベントの登録を行います。 時間特有または時間に基づいたポリシーをトリガーする必要があるときに、この Tcl コマンド拡張を使用します。このイベントタイマーは、イベントのパブリッシャとサブスクライバの両方です。パブリッシャの部分は、名前付きタイマーがオフになるという条件を示します。サブスクライバの部分は、イベントが登録されているタイマーの名前を示します。

（注） CRON および絶対時間の指定は、現地時間で動作します。

構文

```
event_register_timer watchdog|countdown|absolute|cron
[name ?] [cron_entry ?]
[time ?]
[queue_priority low|normal|high] [maxrun ?]
[nice 0|1]
```

引数

watchdog	（必須）ウォッチドッグ タイマー。
countdown	（必須）カウントダウン タイマー。
absolute	（必須）絶対タイマー。
cron	（必須）CRON タイマー。
name	（任意）タイマーの名前。

cron_entry	

（任意）CRON タイマータイプが指定される場合に、エントリを指定する必要があります。他のいずれかのタイマータイプが指定される場合には、指定しないでください。 cron_entry は、UNIX CRON デーモンで使用される部分的な UNIX Crontab エントリ（最初の 5 つのフィールド）です。

cron_entry の指定は、5 つのフィールドが使用されるテキスト文字列で構成されます。フィールドは、空白文字で区切られます。フィールドは、CRON タイマーイベントがトリガーされるときの時刻と日付を表します。フィールドは、表 16：CRON イベントがトリガーされるときの時刻と日付，（87 ページ） で説明されています。

番号の範囲を使用できます。範囲は、ハイフンで区切られる 2 つの数字で表示されます。範囲には、2 つの数字自身も含まれます。 たとえば、時刻に入力される 8-11 は、8 時、9 時、10 時、および 11 時での実行を示します。

フィールドはアスタリスク記号（*）も使用でき、これは常に「first-last」を表します。

リストを使用できます。リストは、カンマで区切られた番号のセット（または範囲）です。例："1,2,5,9" および "0-4,8-12"。

手順の値は、範囲の組み合わせで使用できます。 範囲に続く「/<number>」によって、範囲内での省略値を指定します。 たとえば、2 時間ごとにイベントのトリガーを指定する場合、「0-23/2」を hour フィールドに使用します。アスタリスク記号後にも手順を使用でき、「2 時間ごと」と指定する場合は、「*/2」を使用します。

month フィールドと day of week フィールドには、名前も使用できます。特定の日または月の最初の 3 文字を使用します（ケースは問題ではありません）。名前の範囲またはリストは使用できません。

タイマー イベントがトリガーされる日は、day of month と day of week の 2 つのフィールドで指定できます。 両方のフィールドが制限される（つまり * ではない）場合、いずれかのフィー

	ルドが現在の時刻と一致すると、イベントがトリガーされます。 たとえば、「30 4 1,15 * 5」の場合、各月の 1 日と 15 日に加え、金曜日の午前 4:30 にイベントがトリガーされます。 最初の 5 つのフィールドの代わりに、7 つの特殊文字列の 1 つが表示されることがあります。これらの 7 つの特殊文字列は、表 17：cron_entry の特殊文字列, （88 ページ） で説明します。 例 1：「0 0 1,15 * 1」では、各月の 1 日と 15 日、および月曜日ごとに、真夜中の 0 時に、イベントがトリガーされます。 1 つのフィールドによってのみ日を指定する場合、他のフィールドは * に設定する必要があります。「0 0 * * 1」では、月曜日にのみ、真夜中の 0 時に、イベントがトリガーされます。 例 2：「15 16 1 * *」では、各月の 1 日の午後 4:15 にイベントがトリガーされます。 例 3：「0 12 * * 1-5」では、各週の月曜日から金曜日まで、正午に、イベントがトリガーされます。 例 4：「@weekly」では、1 週間に一度、日曜日の真夜中の 0 時に、イベントがトリガーされます。
time	（任意）CRON 以外のタイマータイプが指定される場合に、時刻を指定する必要があります。 CRON タイマータイプが指定される場合には、指定しないでください。 ウォッチドッグ タイマーとカウントダウン タイマーでは、タイマーの期限が切れるまでの秒およびミリ秒の単位での数です。絶対タイマーでは、期限切れ時刻のカレンダー時間です。 時間は、SSSSSSSSSS[.MMM] 形式で指定します。 SSSSSSSSSS は、0 ～ 4294967295 の秒数を表す整数で、MMM は 0 ～ 999 のミリ秒数を表す整数である必要があります。 期限の絶対日付は、1970 年 1 月 1 日以降の秒およびミリ秒の単位での数です。 指定された日付がすでに過ぎた場合、タイマーの期限はただちに切れます。

queue_priority	（任意）スクリプトのキューイングに使用されるプライオリティ レベル。normal は、low プライオリティよりも高く、high プライオリティよりも低いプライオリティです。 このプライオリティは実行プライオリティではなく、キューイング プライオリティです。 この引数が指定されない場合、デフォルトのプライオリティは normal です。
maxrun	（任意）スクリプトの最大ランタイム（SSSSSSSSSS[.MMM] 形式で指定します。SSSSSSSSSS は、0 ～ 4294967295 の秒数を表す整数で、MMM は 0 ～ 999 のミリ秒数を表す整数である必要があります）。 この引数が指定されない場合、デフォルトの 20 秒ランタイム制限が使用されます。
nice	（任意）ポリシー実行時間のプライオリティ設定。 *nice* 引数が 1 に設定されている場合、ポリシーは、デフォルト プライオリティよりも低い実行時プライオリティで実行されます。デフォルト値は 0 です。

表 *16*：*CRON* イベントがトリガーされるときの時刻と日付

フィールド	使用可能な値
minute	0 ～ 59
hour	0 ～ 23
day of month	1 ～ 31
month	1 ～ 12（または名前、表 17：cron_entry の特殊文字列, （88 ページ）を参照）
day of week	0 ～ 7（0 または 7 が日曜日、または名前。表 17：cron_entry の特殊文字列, （88 ページ） を参照）

表 *17* : *cron_entry* の特殊文字列

文字列	意味
@yearly	1 年に 1 回トリガーする、「0 0 1 1 *」。
@annually	@yearly と同じ。
@monthly	1 か月に 1 回トリガーする、「0 0 1 * *」。
@weekly	1 週間に 1 回トリガーする、「0 0 * * 0」。
@daily	1 日に 1 回トリガーする、「0 0 * * *」。
@midnight	@daily と同じ。
@hourly	1 時間に 1 回トリガーする、「0 * * * *」。

結果文字列

なし

_cerrno を設定

No

関連項目

event_register_timer_subscriber, （88 ページ）

event_register_timer_subscriber

サブスクライバとしてタイマー イベントの登録を行います。 この Tcl コマンド拡張を使用すると、サブスクライバとして、登録するイベント タイマーの名前が指定されます。 イベント タイマーは、別のポリシーまたは別のプロセスに依存して、カウンタが実際に操作されます。 たとえば、policyB はタイマー加入者ポリシーとして動作しますが、policyA（タイマー ポリシーは不要ですが）では、register_counter、timer_arm、または timer_cancel の各 Tcl コマンド拡張を使用して、policyB で参照されているカウンタが操作されます。

構文

```
event_register_timer_subscriber watchdog|countdown|absolute|cron
name ? [queue_priority low|normal|high] [maxrun ?] [nice 0|1]
```

引数

watchdog	（必須）ウォッチドッグ タイマー。

countdown	（必須）カウントダウン タイマー。
absolute	（必須）絶対タイマー。
cron	（必須）CRON タイマー。
name	（必須）タイマーの名前。
queue_priority	（任意）スクリプトのキューイングに使用されるプライオリティ レベル。normal は、low プライオリティよりも高く、high プライオリティよりも低いプライオリティです。 このプライオリティは実行プライオリティではなく、キューイング プライオリティです。 この引数が指定されない場合、デフォルトのプライオリティは normal です。
maxrun	（任意）スクリプトの最大ランタイム（SSSSSSSSSS[.MMM] 形式で指定します。SSSSSSSSSS は、0 ～ 4294967295 の秒数を表す整数で、MMM は 0 ～ 999 のミリ秒数を表す整数である必要があります）。 この引数が指定されない場合、デフォルトの 20 秒ランタイム制限が使用されます。
nice	（任意）ポリシー実行時間のプライオリティ設定。 nice 引数が 1 に設定されている場合、ポリシーは、デフォルトプライオリティよりも低い実行時プライオリティで実行されます。デフォルト値は 0 です。

（注） タイマーイベントまたはカウンタイベントの登録を行う EEM ポリシーは、パブリッシャとサブスクライバの両方として動作できます。

結果文字列

なし

_cerrno を設定

No

関連項目

event_register_timer, （83 ページ）

event_register_wdsysmon

Watchdog System Monitor イベントの登録を行います。 この Tcl コマンド拡張を使用すると、いくつかのサブイベントまたは条件の組み合わせである複合イベントの登録が行われます。 たとえば、**event_register_wdsysmon** コマンドを使用して、特定処理の CPU の使用率が 80% を超え、かつ、処理に使用されるメモリが、その初期割り当ての 50% よりも大きいときの、条件の組み合わせの登録を行うことができます。 この Tcl コマンド拡張は、ソフトウェア モジュール方式イメージでのみサポートされます。

構文

```
event_register_wdsysmon [timewin ?]
[sub12_op and|or|andnot]
[sub23_op and|or|andnot]
[sub34_op and|or|andnot]
[sub1 subevent-description]
[sub2 subevent-description]
[sub3 subevent-description]
[sub4 subevent-description] [node ?]
[queue_priority low|normal|high]
[maxrun ?] [nice 0|1]
```

引数

timewin	（任意）イベントが生成されるようにするために、すべてのサブイベントが発生する必要がある時間ウィンドウを SSSSSSSSSS[.MMM] 形式で指定します。 SSSSSSSSSS 形式は、0 ～ 4294967295 の秒数を表す整数である必要があります。 MMM 形式は、0 ～ 999 のミリ秒数を表す整数である必要があります。
sub12_op	（任意）サブイベント 1 とサブイベント 2 とを比較する組み合わせ演算子。
sub34_op	（任意）サブイベント 1、2、サブイベント 3、サブイベント 4 とを比較する組み合わせ演算子。
sub1	（任意）サブイベント 1 を指定します。
subevent-description	（任意）サブイベントの構文。
sub2	（任意）サブイベント 2 を指定します。
sub3	（任意）サブイベント 3 を指定します。

sub4	（任意）サブイベント 4 を指定します。
node	（任意）デッドロック条件がモニタされるノード名は、「node」という語句と、それに続く、次の形式を使用してスラッシュ（/）で区切られた 2 つのフィールドで構成される、文字列です。 `node<slot-number>/<cpu-number>` slot-number は、ハードウェア スロット番号です。 cpu-number は、ハードウェア CPU 番号です。 たとえば、スロット 0 にある Cisco Catalyst 6500 シリーズ スイッチのスーパーバイザ カードの SP CPU は、node0/0 と指定されます。 たとえば、スロット 0 にある Cisco Catalyst 6500 シリーズ スイッチのスーパーバイザ カードの RP CPU は、node0/1 と指定されます。 node 引数が指定されない場合、デフォルトのノード指定は、登録が行われているローカル ノードです。
queue_priority	（任意）スクリプトのキューイングに使用されるプライオリティ レベル。normal は、low プライオリティよりも高く、high プライオリティよりも低いプライオリティです。 このプライオリティは実行プライオリティではなく、キューイング プライオリティです。 この引数が指定されない場合、デフォルトのプライオリティは normal です。
maxrun	（オプション）SSSSSSSSSS[.MMM] 形式で指定される最大実行時間。 SSSSSSSSSS 形式は、0 ～ 4294967295 の秒数を表す整数である必要があります。 MMM 形式は、0 ～ 999 のミリ秒数を表す整数である必要があります。 この引数が指定されない場合、デフォルトの 20 秒ランタイム制限が使用されます。
nice	（任意）ポリシー実行時間のプライオリティ設定。 *nice* 引数が 1 に設定されている場合、ポリシーは、デフォルトプライオリティよりも低い実行時プライオリティで実行されます。デフォルト値は 0 です。

サブイベント

subevent description の構文は、7 つのケースのうちの 1 つを使用できます。

subevent descriptions の引数では、number 引数の値に次の制約事項が適用されます。

- dispatch_mgr では、val は、0 ～ 4294967295 の範囲の整数である必要があります。
- cpu_proc および cpu_tot では、val は、0 ～ 100 の整数である必要があります。
- mem_proc、mem_tot_avail、および mem_tot_used では、is_percent が偽の場合、val は、0 ～ 4294967295 の範囲の整数である必要があります。

1 deadlock procname ?

引数

procname	（必須）デッドロック条件をモニタするプロセス名を指定する正規表現。 指定された場合、サブイベントによって、時間ウィンドウは無視されます。

1 dispatch_mgr [procname ?] [op gt|ge|eq|ne|lt|le] [val ?] [period ?]

引数

procname	（任意）dispatch_manager ステータスをモニタするプロセス名を指定する正規表現。
op	（任意）収集したイベント数を指定した値と比較するために使用する比較演算子。 真の場合、イベントが発生します。
val	（任意）発生したイベントの数を比較する値。
period	（任意）発生したイベント数の時間（SSSSSSSSSS[.MMM] 形式で指定します。 SSSSSSSSSS 形式は、0 ～ 4294967295 の秒数を表す整数である必要があります。 MMM 形式は、0 ～ 999 のミリ秒数を表す整数である必要があります。 この引数が指定されない場合は、最新のサンプルが使用されます。

1 cpu_proc [procname ?] [op gt|ge|eq|ne|lt|le] [val ?] [period ?]

引数

procname	（任意）CPUの使用条件をモニタするプロセス名を指定する正規表現。
op	（任意）収集した CPU 使用率サンプル パーセンテージを指定したパーセンテージ値と比較するために使用する比較演算子。 真の場合、イベントが発生します。
val	（任意）サンプル期間中の平均 CPU 使用率を比較するパーセンテージ値。
period	（任意）サンプルの収集を平均するための時間を SSSSSSSSSS[.MMM] 形式で指定します。 SSSSSSSSSS 形式は、0 ～ 4294967295 の秒数を表す整数である必要があります。 MMM 形式は、0 ～ 999 のミリ秒数を表す整数である必要があります。 この引数が指定されない場合は、最新のサンプルが使用されます。

1 cpu_tot [op gt|ge|eq|ne|lt|le] [val ?] [period ?]

引数

op	（任意）収集した合計システム CPU 使用率サンプル パーセンテージを指定したパーセンテージ値と比較するために使用する比較演算子。 真の場合、イベントが発生します。
val	（任意）サンプル期間中の平均 CPU 使用率を比較するパーセンテージ値。
period	（任意）サンプルの収集を平均するための時間を SSSSSSSSSS[.MMM] 形式で指定します。 SSSSSSSSSS 形式は、0 ～ 4294967295 の秒数を表す整数である必要があります。 MMM 形式は、0 ～ 999 のミリ秒数を表す整数である必要があります。 この引数が指定されない場合は、最新のサンプルが使用されます。

1 mem_proc [procname ?] [op gt|ge|eq|ne|lt|le] [val ?] [is_percent TRUE|FALSE] [period ?]

引数

procname	（任意）メモリ使用状況をモニタするプロセス名を指定する正規表現。
op	（任意）収集した使用メモリを指定した値と比較するために使用する比較演算子。 真の場合、イベントが発生します。
val	（任意）キロバイト単位で指定される絶対値、またはパーセンテージ。 パーセンテージは、指定された時間内で最も古いサンプルと、最新のサンプルとの違いを表します。 メモリ使用量が時間内で 150 KB から 300 KB に増えた場合、増加パーセンテージは 100 です。 この値と測定された値が比較されます。
is_percent	（任意）TRUE に設定されている場合、パーセンテージの値が収集され、比較されます。 これ以外の場合、絶対値が収集され、比較されます。
period	（任意）is_percent が TRUE に設定される場合、時間のパーセンテージが計算されます。 これ以外の場合、収集サンプルの期間は平均化され、SSSSSSSSSS[.MMM] 形式で指定されます。 SSSSSSSSSS 形式は、0 ～ 4294967295 の秒数を表す整数である必要があります。 MMM 形式は、0 ～ 999 のミリ秒数を表す整数である必要があります。 この引数が指定されない場合は、最新のサンプルが使用されます。

1 mem_tot_avail [op gt|ge|eq|ne|lt|le] [val ?] [is_percent TRUE|FALSE] [period ?]

引数

op	（任意）収集した使用可能メモリを指定した値と比較するために使用する比較演算子。 真の場合、イベントが発生します。

val	（任意）キロバイト単位で指定される絶対値、またはパーセンテージ。パーセンテージは、指定された時間内で最も古いサンプルと、最新のサンプルとの違いを表します。使用可能なメモリ使用量が時間内で 300 KB から 150 KB に減った場合、減少パーセンテージは 50 です。 この値と測定された値が比較されます。
is_percent	（任意）TRUE に設定されている場合、パーセンテージの値が収集され、比較されます。 これ以外の場合、絶対値が収集され、比較されます。
period	（任意）is_percent が TRUE に設定される場合、時間のパーセンテージが計算されます。 これ以外の場合、収集サンプルの期間は平均化され、SSSSSSSSSS[.MMM] 形式で指定されます。SSSSSSSSSS 形式は、0 ～ 4294967295 の秒数を表す整数である必要があります。 MMM 形式は、0 ～ 999 のミリ秒数を表す整数である必要があります。 この引数が指定されない場合は、最新のサンプルが使用されます。

1 mem_tot_used [op gt|ge|eq|ne|lt|le] [val ?] [is_percent TRUE|FALSE] [period ?]

引数

op	（任意）収集した使用メモリを指定した値と比較するために使用する比較演算子。 真の場合、イベントが発生します。
val	（任意）キロバイト単位で指定される絶対値、またはパーセンテージ。パーセンテージは、指定された時間内で最も古いサンプルと、最新のサンプルとの違いを表します。メモリ使用量が時間内で 150 KB から 300 KB に増えた場合、増加パーセンテージは 100 です。 この値と測定された値が比較されます。
is_percent	（任意）TRUE に設定されている場合、パーセンテージの値が収集され、比較されます。 これ以外の場合、絶対値が収集され、比較されます。

period	（任意）is_percent が TRUE に設定される場合、時間のパーセンテージが計算されます。 これ以外の場合、収集サンプルの期間は平均化され、SSSSSSSSSS[.MMM] 形式で指定されます。 SSSSSSSSSS 形式は、0 ～ 4294967295 の秒数を表す整数である必要があります。 MMM 形式は、0 ～ 999 のミリ秒数を表す整数である必要があります。 この引数が指定されない場合は、最新のサンプルが使用されます。 （注） is_percent が真に設定されている場合、この引数は必須です。これ以外の場合、この引数は任意です。

結果文字列

なし

_cerrno を設定

No

（注） サブイベントの説明内部では、各引数は、位置に依存しません。

Embedded Event Manager イベント情報 Tcl コマンド拡張

次の EEM イベント情報 Tcl コマンド拡張がサポートされています。

event_reqinfo

現在のポリシーを実行させる原因のイベントについての情報を問い合わせます。

構文

```
event_reqinfo
```

引数

なし

結果文字列

ポリシーが正常に実行される場合、ポリシーをトリガーするイベントの特性が戻されます。 次の項では、各イベント ディテクタで戻される特性を示します。

EEM_EVENT_APPLICATION について

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"sub_system 0x%x type %u data1 {%s} data2 {%s} data3 {%s} data4 {%s}"
```

イベント タイプ	説明
event_id	パブリッシュされた該当イベントの ID を示す一意の番号。同一のイベントで複数のポリシーを実行可能であり、その場合、各ポリシーは同一の event_id を保持します。
event_type	イベントのタイプ。
event_type_string	このイベント タイプのイベントの名前を表す ASCII 文字列。
event_pub_secevent_pub_msec	イベントが Embedded Event Manager にパブリッシュされたときの、秒単位およびミリ秒単位の時間。
sub_system	アプリケーションイベントをパブリッシュした EEM ポリシーに割り当てられる番号。 他のすべての番号は Cisco での使用のために予約されているため、番号は 798 に設定されます。
type	指定されたコンポーネント内のイベント サブタイプ。
data1data2data3data4	イベントがパブリッシュされるときに、アプリケーション固有のイベントに渡される、引数データ。データは、文字テキスト、環境変数、または、この 2 つの組み合わせです。

EEM_EVENT_COUNTER について

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u"
"name {%s}"
```

イベント タイプ	説明
event_id	パブリッシュされた該当イベントの ID を示す一意の番号。同一のイベントで複数のポリシーを実行可能であり、その場合、各ポリシーは同一の event_id を保持します。
event_type	イベントのタイプ。

イベント タイプ	説明
event_type_string	このイベント タイプのイベントの名前を表す ASCII 文字列。
event_pub_secevent_pub_msec	イベントが Embedded Event Manager にパブリッシュされたときの、秒単位およびミリ秒単位の時間。
name	カウンタ名。

EEM_EVENT_NONE について

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
```

イベント タイプ	説明
event_id	パブリッシュされた該当イベントの ID を示す一意の番号。同一のイベントで複数のポリシーを実行可能であり、その場合、各ポリシーは同一の event_id を保持します。
event_type	イベントのタイプ。
event_type_string	このイベント タイプのイベントの名前を表す ASCII 文字列。
event_pub_secevent_pub_msec	イベントが Embedded Event Manager にパブリッシュされたときの、秒単位およびミリ秒単位の時間。

EEM_EVENT_OIR について

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"slot %u event %s"
```

イベント タイプ	説明
event_id	パブリッシュされた該当イベントの ID を示す一意の番号。同一のイベントで複数のポリシーを実行可能であり、その場合、各ポリシーは同一のイベント ID を保持します。
event_type	イベントのタイプ。

イベント タイプ	説明
event_type_string	このイベント タイプのイベントの名前を表す ASCII 文字列。
event_pub_secevent_pub_msec	イベントが Embedded Event Manager にパブリッシュされたときの、秒単位およびミリ秒単位の時間。
slot	影響が及ぼされるカードのスロット番号。
event	OIR の削除イベントまたは OIR の挿入イベントを表す、removed または online の文字列を示します。

EEM_EVENT_PROCESS について（ソフトウェアモジュール方式のみ）

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"sub_system 0x%x instance %u process_name {%s} path {%s} exit_status 0x%x"
"respawn_count %u last_respawn_sec %ld last_respawn_msec %ld fail_count %u"
"dump_count %u node_name {%s}"
```

イベント タイプ	説明
event_id	パブリッシュされた該当イベントの ID を示す一意の番号。同一のイベントで複数のポリシーを実行可能であり、その場合、各ポリシーは同一の event_id を保持します。
event_type	イベントのタイプ。
event_type_string	このイベント タイプのイベントの名前を表す ASCII 文字列。
event_pub_secevent_pub_msec	イベントが Embedded Event Manager にパブリッシュされたときの、秒単位およびミリ秒単位の時間。
sub_system	アプリケーション固有のイベントをパブリッシュした EEM ポリシーに割り当てられる番号。他のすべての番号は Cisco での使用のために予約されているため、番号は 798 に設定されます。
instance	プロセス インスタンス ID。

イベント タイプ	説明
process_name	プロセス名。
path	パスを含むプロセスの絶対名。
exit_status	プロセスの最後の終了ステータス。
respawn_count	プロセスが再起動された回数。
last_respawn_seclast_respawn_msec	最後の再起動が発生したカレンダー時間。
fail_count	失敗したプロセスの再起動試行の回数。プロセスが正常に再起動されると、0 にリセットされます。
イベント タイプ	**説明**
dump_count	プロセスで取られたコア ダンプの数。
node_name	プロセスが存在するノードの名前。 ノード名は、「node」という語句と、それに続く、次の形式を使用してスラッシュ文字で区切られた 2 つのフィールドで構成される、文字列です。 node<slot-number>/<cpu-number> slot-number は、ハードウェア スロット番号です。 cpu-number は、ハードウェア CPU 番号です。

EEM_EVENT_RF について

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u"
"event {%s}"
```

イベント タイプ	説明
event_id	パブリッシュされた該当イベントの ID を示す一意の番号。同一のイベントで複数のポリシーを実行可能であり、その場合、各ポリシーは同一の event_id を保持します。
event_type	イベントのタイプ。
event_type_string	このイベント タイプのイベントの名前を表す ASCII 文字列。

イベント タイプ	説明
event_pub_secevent_pub_msec	イベントが Embedded Event Manager にパブリッシュされたときの、秒単位およびミリ秒単位の時間。
event	このイベントが発生する原因となる RF の進行またはステータス イベント通知。

EEM_EVENT_SYSLOG_MSG について

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"msg {%s}"
```

イベント タイプ	説明
event_id	パブリッシュされた該当イベントの ID を示す一意の番号。同一のイベントで複数のポリシーを実行可能であり、その場合、各ポリシーは同一の event_id を保持します。
event_type	イベントのタイプ。
event_type_string	このイベント タイプのイベントの名前を表す ASCII 文字列。
event_pub_secevent_pub_msec	イベントが Embedded Event Manager にパブリッシュされたときの、秒単位およびミリ秒単位の時間。
msg	パターンと一致する最後の Syslog メッセージ。

EEM_EVENT_TIMER_ABSOLUTE

EEM_EVENT_TIMER_COUNTDOWN

EEM_EVENT_TIMER_WATCHDOG について

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"timer_type %s timer_time_sec %ld timer_time_msec %ld"
"timer_remain_sec %ld timer_remain_msec %ld"
```

イベント タイプ	説明
event_id	パブリッシュされた該当イベントの ID を示す一意の番号。同一のイベントで複数のポリシーを実行可能であり、その場合、各ポリシーは同一の event_id を保持します。
event_type	イベントのタイプ。
event_type_string	このイベント タイプのイベントの名前を表す ASCII 文字列。
event_pub_secevent_pub_msec	イベントが Embedded Event Manager にパブリッシュされたときの、秒単位およびミリ秒単位の時間。
timer_type	タイマーのタイプ。 次のいずれかです。 • watchdog • countdown • absolute
timer_time_sectimer_time_msec	タイマーの期限が切れる時間。
timer_remain_sectimer_remain_msec	次の期限切れ前の残りの時間。

EEM_EVENT_TIMER_CRON について

```
"event_id %u event_type %u event_type_string {%s} event_pub_sec %u event_pub_msec %u"
"timer_type {%s} timer_time_sec %ld timer_time_msec %ld"
```

イベント タイプ	説明
event_id	パブリッシュされた該当イベントの ID を示す一意の番号。同一のイベントで複数のポリシーを実行可能であり、その場合、各ポリシーは同一の event_id を保持します。
event_type	イベントのタイプ。
event_type_string	このイベント タイプのイベントの名前を表す ASCII 文字列。

イベント タイプ	説明
event_pub_secevent_pub_msec	イベントが Embedded Event Manager にパブリッシュされたときの、秒単位およびミリ秒単位の時間。
timer_type	タイマーのタイプ。
timer_time_sectimer_time_msec	タイマーの期限が切れる時間。

EEM_EVENT_TRACK について

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u"
"track_number {%u} track_state {%s}"
```

イベント タイプ	説明
event_id	パブリッシュされた該当イベントの ID を示す一意の番号。同一のイベントで複数のポリシーを実行可能であり、その場合、各ポリシーは同一のイベント ID を保持します。
event_type	イベントのタイプ。
event_type_string	このイベント タイプのイベントの名前を表す ASCII 文字列。
event_pub_secevent_pub_msec	イベントが Embedded Event Manager にパブリッシュされたときの、秒単位およびミリ秒単位の時間。
track_number	イベントがトリガーされる原因となるトラックされるオブジェクトの番号。
track_state	イベントがトリガーされたときのトラックされるオブジェクトの状態。有効な値は up または down です。

EEM_EVENT_WDSYSMON について

```
"event_id %u event_type %u event_type_string {%s} %u event_pub_sec %u event_pub_msec %u"
"num_subs %u"
```

イベント タイプ	説明
event_id	パブリッシュされた該当イベントの ID を示す一意の番号。同一のイベントで複数のポリシーを実行可能であり、その場合、各ポリシーは同一の event_id を保持します。
event_type	イベントのタイプ。
event_type_string	このイベント タイプのイベントの名前を表す ASCII 文字列。
event_pub_secevent_pub_msec	イベントが Embedded Event Manager にパブリッシュされたときの、秒単位およびミリ秒単位の時間。
num_subs	サブイベント番号。

サブイベント情報文字列は、次のような、デッドロック サブイベント用です。

```
"{type %s num_entries %u entries {entry 1, entry 2, ...}}"
```

サブイベント タイプ	説明
type	Wdsysmon サブイベントのタイプ。
num_entries	デッドロックのプロセスおよびスレッドの番号。
entries	デッドロックのプロセスおよびスレッドの情報。

各エントリは次のとおりです。

```
"{node {%s} procname {%s} pid %u tid %u state %s b_node %s b_procname %s b_pid %u
b_tid %u}"
```

このエントリでは、プロセス A のスレッド m によって、プロセス B のスレッド n でブロックされるシナリオが記述されているとすると、次のようになります。

サブイベント タイプ	説明
node	プロセス A のスレッド m があるノードの名前。
procname	プロセス A の名前。

サブイベント タイプ	説明
pid	プロセス A のプロセス ID。
tid	プロセス A のスレッド m のスレッド ID。
state	プロセス A のスレッド m のスレッドの状態。次のいずれかです。 • STATE_CONDVAR • STATE_DEAD • STATE_INTR • STATE_JOIN • STATE_MUTEX • STATE_NANOSLEEP • STATE_READY • STATE_RECEIVE • STATE_REPLY • STATE_RUNNING • STATE_SEM • STATE_SEND • STATE_SIGSUSPEND • STATE_SIGWAITINFO • STATE_STACK • STATE_STOPPED • STATE_WAITPAGE • STATE_WAITTHREAD
b_node	プロセス B のスレッドがあるノードの名前。
b_procname	プロセス B の名前。
b_pid	プロセス B のプロセス ID。
b_tid	プロセス B のスレッド n のスレッド ID。0 は、プロセス A のスレッド m は、プロセス B のすべてのスレッド上でブロックされることを意味します。

dispatch_mgr サブイベントについて

```
"{type %s node {%s} procname {%s} pid %u value %u sec %ld msec %ld}"
```

サブイベント タイプ	説明
type	Wdsysmon サブイベントのタイプ。
node	POSIX プロセスが存在するノードの名前。
procname	このサブイベントの POSIX プロセス名。
pid	このサブイベントの POSIX プロセス ID。 （注） 前述の3つのフィールドは、このディスパッチ マネージャのオーナー プロセスについて説明します。
value	sec 変数と msec 変数が、0 に指定されるか、または、イベント登録 Tcl コマンド拡張で指定されない場合、ディスパッチマネージャによって処理されるイベント数は、最新のサンプルにあります。 時間ウィンドウが指定され、登録 Tcl コマンド拡張でゼロよりも大きい場合、このディスパッチマネージャによって処理されるイベントの合計数は、該当する時間ウィンドウにあります。
secmsec	イベント登録 Tcl コマンド拡張で、sec 変数と msec 変数が 0 に指定されているか、または指定されていない場合、両方とも 0 です。 登録 Tcl コマンド拡張で時間ウィンドウが指定され、かつその時間ウィンドウがゼロよりも大きい場合、sec 変数および msec 変数は、この時間ウィンドウの最も古いサンプルと最新のサンプルとの実際の時間の差分です。

cpu_proc サブイベントについて

```
"{type %s node {%s} procname {%s} pid %u value %u sec %ld msec %ld}"
```

サブイベント タイプ	説明
type	Wdsysmon サブイベントのタイプ。
node	POSIX プロセスが存在するノードの名前。

サブイベント タイプ	説明
procname	このサブイベントの POSIX プロセス名。
pid	このサブイベントの POSIX プロセス ID。 （注） 前述の3つのフィールドは、そのCPU使用率がモニタされているプロセスについて説明します。
value	sec 変数と msec 変数が、0 に指定されるか、または、イベント登録 Tcl コマンド拡張で指定されない場合、プロセス CPU 使用率は、最新のサンプルにあります。時間ウィンドウが指定され、登録 Tcl コマンド拡張でゼロよりも大きい場合、プロセス CPU 使用率の平均は、該当する時間ウィンドウにあります。
secmsec	イベント登録 Tcl コマンド拡張で、sec 変数と msec 変数が 0 に指定されているか、または指定されていない場合、両方とも 0 です。 登録 Tcl コマンド拡張で時間ウィンドウが指定され、かつその時間ウィンドウがゼロよりも大きい場合、sec 変数および msec 変数は、この時間ウィンドウの最も古いサンプルと最新のサンプルとの実際の時間の差分です。

cpu_tot サブイベントについて

```
"{type %s node {%s} value %u sec %ld msec %ld}"
```

サブイベント タイプ	説明
type	Wdsysmon サブイベントのタイプ。
node	CPU使用率の合計がモニタされているノードの名前。
value	sec 変数と msec 変数が、0 に指定されるか、または、イベント登録 Tcl コマンド拡張で指定されない場合、合計 CPU 使用率は、最新のサンプルにあります。時間ウィンドウが指定され、登録Tclコマンド拡張でゼロよりも大きい場合、合計 CPU 使用率の平均は、該当する時間ウィンドウにあります。

サブイベント タイプ	説明
secmsec	イベント登録 Tcl コマンド拡張で、sec 変数と msec 変数が 0 に指定されているか、または指定されていない場合、両方とも 0 です。 登録 Tcl コマンド拡張で時間ウィンドウが指定され、かつその時間ウィンドウがゼロよりも大きい場合、sec 変数および msec 変数は、この時間ウィンドウの最も古いサンプルと最新のサンプルとの実際の時間の差分です。

mem_proc サブイベントについて

```
"{type %s node {%s} procname {%s} pid %u is_percent %s value %u diff %d sec %ld msec %ld}"
```

サブイベント タイプ	説明
type	Wdsysmon サブイベントのタイプ。
node	POSIX プロセスが存在するノードの名前。
procname	このサブイベントの POSIX プロセス名。
pid	このサブイベントの POSIX プロセス ID。 （注） 前述の 3 つのフィールドは、そのメモリ使用率がモニタされているプロセスについて説明します。
is_percent	TRUE または FALSE のいずれかです。 TRUE は、値がパーセント値であることを示します。FALSE は、値が絶対値であることを示します（平均値の場合もあります）。
value	sec 変数と msec 変数が、0 に指定されるか、または、イベント登録 Tcl コマンド拡張で指定されない場合、プロセスで使用されたメモリは、最新のサンプルにあります。時間ウィンドウが指定され、登録 Tcl コマンド拡張でゼロよりも大きい場合、プロセスで使用されたメモリ使用率の平均は、該当する時間ウィンドウにあります。
サブイベント タイプ	説明

サブイベント タイプ	説明
diff	sec 変数と msec 変数が、0 に指定されるか、または、イベント登録 Tcl コマンド拡張で指定されない場合、diff は、今まで収集された、プロセスで使用された最初のメモリ サンプルと、プロセスで使用された最新のメモリ サンプルとのパーセンテージによる差分です。時間ウィンドウが指定され、登録 Tcl コマンド拡張でゼロよりも大きい場合、diff は、指定された時間ウィンドウで、最も古いプロセスで使用されたメモリ使用率と最新のプロセスで使用されたメモリ使用率とのパーセンテージによる差分です。
secmsec	イベント登録 Tcl コマンド拡張で、sec 変数と msec 変数が 0 に指定されているか、または指定されていない場合、両方とも 0 です。 登録 Tcl コマンド拡張で時間ウィンドウが指定され、かつその時間ウィンドウがゼロよりも大きい場合、sec 変数および msec 変数は、この時間ウィンドウの最も古いサンプルと最新のサンプルとの実際の時間の差分です。

is_percent 引数が FALSE であり、イベント登録 Tcl コマンド拡張で *sec* 引数と *msec* 引数が 0 に指定されているか、または指定されていない場合は、次のようになります。

- *value* は、最新のサンプルでプロセスによって使用されたメモリです。
- *diff* は 0 です。
- *sec* と *msec* は、両方とも 0 です。

is_percent 引数が FALSE であり、イベント登録 Tcl コマンド拡張で時間ウィンドウがゼロよりも大きい値に指定されている場合は、次のようになります。

- *value* は、指定された時間ウィンドウでプロセスによって使用されたメモリ サンプル値の平均です。
- *diff* は 0 です。
- *sec* および *msec* は、両方とも、この時間ウィンドウの、最も古いタイム スタンプと最新のタイム スタンプとの実際の時間の差分です。

is_percent 引数が TRUE であり、イベント登録 Tcl コマンド拡張で時間ウィンドウがゼロよりも大きい値に指定されている場合は、次のようになります。

- value は 0 です。

- *diff* は、指定された時間ウィンドウの、最も古いプロセスで使用されたメモリ サンプルと最新のプロセスで使用されたメモリ サンプルとのパーセンテージによる差分です。
- *sec* および **msec** は、この時間ウィンドウの、最も古いプロセスで使用されたメモリ サンプルと最新のプロセスで使用されたメモリ サンプルとの実際の時間の差分です。

is_percent 引数が TRUE であり、イベント登録 Tcl コマンド拡張で *sec* 引数と *msec* 引数が 0 に指定されているか、または指定されていない場合は、次のようになります。

- *value* は 0 です。
- *diff* は、今まで収集された、プロセスで使用された最新のメモリ サンプルと、プロセスで使用された最新のメモリ サンプルとのパーセンテージによる差分です。
- *sec* および *msec* は、今まで収集された、プロセスで使用された最初のメモリ サンプルとプロセスで使用された最新のメモリ サンプルとの実際の時間の差分です。

mem_tot_avail サブイベントについて

```
"{type %s node {%s} is_percent %s used %u avail %u diff %d sec %ld msec %ld}"
```

サブイベント タイプ	説明
type	Wdsysmon サブイベントのタイプ。
node	使用可能なメモリの合計がモニタされているノードの名前。
is_percent	TRUE または FALSE のいずれかです。 TRUE は、値がパーセント値であることを示します。FALSE は、値が絶対値であることを示します（平均値の場合もあります）。
used	sec 変数と msec 変数が、0 に指定されるか、または、イベント登録 Tcl コマンド拡張で指定されない場合、使用されたメモリの合計は、最新のサンプルにあります。時間ウィンドウが指定され、登録 Tcl コマンド拡張でゼロよりも大きい場合、使用された合計メモリ使用率の平均は、該当する時間ウィンドウにあります。
avail	sec 変数と msec 変数が、0 に指定されるか、または、イベント登録 Tcl コマンド拡張で指定されない場合、availは、最新の使用可能なメモリ サンプルの合計にあります。時間ウィンドウが指定され、登録 Tcl コマンド拡張でゼロよりも大きい場合、availは、指定された時間ウィンドウで使用可能なメモリ使用率の合計です。

サブイベント タイプ	説明
diff	sec 変数と msec 変数が、0 に指定されるか、または、イベント登録 Tcl コマンド拡張で指定されない場合、diff は、今まで収集された、最初の使用可能なメモリ サンプルの合計と、最新の使用可能なメモリ サンプルの合計とのパーセンテージによる差分です。時間ウィンドウが指定され、登録 Tcl コマンド拡張でゼロよりも大きい場合、diffは、指定された時間ウィンドウで、最も古い使用可能なメモリ使用率と最新の使用可能なメモリ使用率とのパーセンテージによる差分です。
secmsec	イベント登録 Tcl コマンド拡張で、sec 変数と msec 変数が 0 に指定されているか、または指定されていない場合、両方とも 0 です。時間ウィンドウが指定され、登録 Tcl コマンド拡張でゼロよりも大きい場合、これらの変数は、この時間ウィンドウの、最も古いサンプルと最新のサンプルとの実際の時間の差分です。

is_percent 引数が FALSE であり、イベント登録 Tcl コマンド拡張で sec 引数と msec 引数が 0 に指定されているか、または指定されていない場合は、次のようになります。

- *used* は、最新のサンプルで使用されたメモリの合計です。
- *avail* は、最新のサンプルで使用可能なメモリの合計です。
- *diff* は 0 です。
- *sec* と **msec** は、両方とも 0 です。

is_percent 引数が FALSE であり、イベント登録 Tcl コマンド拡張で時間ウィンドウがゼロよりも大きい値に指定されている場合は、次のようになります。

- *used* は 0 です。
- *avail* は、指定された時間ウィンドウで使用可能なメモリ サンプル値の平均です。
- *diff* は 0 です。
- *sec* および *msec* は、両方とも、この時間ウィンドウの、最も古い使用可能なメモリ サンプルの合計のタイム スタンプと最新の使用可能なメモリ サンプルの合計のタイム スタンプとの実際の時間の差分です。

is_percent 引数が TRUE であり、イベント登録 Tcl コマンド拡張で時間ウィンドウがゼロよりも大きい値に指定されている場合は、次のようになります。

- *used* は 0 です。
- *avail* は 0 です。
- *diff* は、指定された時間ウィンドウの、最も古い使用可能なメモリ サンプルの合計と最新の可能なメモリ サンプルの合計とのパーセンテージによる差分です。
- *sec* および *msec* は、両方とも、この時間ウィンドウの、最も古い使用可能なメモリ サンプルの合計のタイム スタンプと最新の使用可能なメモリ サンプルの合計のタイム スタンプとの実際の時間の差分です。

is_percent 引数が TRUE であり、イベント登録 Tcl コマンド拡張で *sec* 引数と *msec* 引数が 0 に指定されているか、または指定されていない場合は、次のようになります。

- *used* は 0 です。
- *avail* は 0 です。
- *diff* は、今まで収集された、最初の使用可能なメモリ サンプルの合計と、最新の使用可能なメモリ サンプルの合計との間の、パーセンテージによる差です。
- *sec* および msec は、今まで収集された、使用可能な最初のメモリ サンプルの合計と使用可能な最新のメモリ サンプルの合計との間の、実際の時間の差です。

mem_tot_used サブイベントについて

```
"{type %s node {%s} is_percent %s used %u avail %u diff %d sec %ld msec %ld}"
```

サブイベント タイプ	説明
type	Wdsysmon サブイベントのタイプ。
node	使用されているメモリの合計がモニタされているノードの名前。
is_percent	TRUE または FALSE のいずれかです。 TRUE は、値がパーセント値であることを示します。FALSE は、値が絶対値であることを示します（平均値の場合もあります）。
used	sec 変数と msec 変数が、0 に指定されるか、または、イベント登録 Tcl コマンド拡張で指定されない場合、使用されたメモリの合計は、最新のサンプルにあります。時間ウィンドウが指定され、登録 Tcl コマンド拡張でゼロよりも大きい場合、使用された合計メモリ使用率の平均は、該当する時間ウィンドウにあります。

サブイベント タイプ	説明
avail	sec 変数と msec 変数が、0 に指定されるか、または、イベント登録 Tcl コマンド拡張で指定されない場合、availは、最新の使用されたメモリ サンプルの合計にあります。時間ウィンドウが指定され、登録 Tcl コマンド拡張でゼロよりも大きい場合、availは、指定された時間ウィンドウで使用されたメモリ使用率の合計です。
diff	sec 変数と msec 変数が、0 に指定されるか、または、イベント登録 Tcl コマンド拡張で指定されない場合、diff は、今まで収集された、プロセスで使用可能な最初のメモリ サンプルと、プロセスで使用可能な最新のメモリ サンプルとのパーセンテージによる差分です。時間ウィンドウが指定され、登録 Tcl コマンド拡張でゼロよりも大きい場合、diff は、指定された時間ウィンドウで、最も古い使用されたメモリ使用率の合計と最新の使用されたメモリ使用率の合計とのパーセンテージによる差分です。
secmsec	イベント登録 Tcl コマンド拡張で、sec 変数と msec 変数が 0 に指定されているか、または指定されていない場合、両方とも 0 です。 登録 Tcl コマンド拡張で時間ウィンドウが指定され、かつその時間ウィンドウがゼロよりも大きい場合、sec 変数および msec 変数は、この時間ウィンドウの最も古いサンプルと最新のサンプルとの実際の時間の差分です。

is_percent 引数が FALSE であり、イベント登録 Tcl コマンド拡張で *sec* 引数と *msec* 引数が 0 に指定されているか、または指定されていない場合は、次のようになります。

- *used* は、最新のサンプルで使用されたメモリの合計です。
- *avail* は、最新のサンプルで使用可能なメモリの合計です。
- *diff* は 0 です。
- *sec* と *msec* は、両方とも 0 です。

is_percent 引数が FALSE であり、イベント登録 Tcl コマンド拡張で時間ウィンドウがゼロよりも大きい値に指定されている場合は、次のようになります。

- *used* は、指定された時間ウィンドウで使用されたメモリ サンプル値の平均です。

- *avail* は 0 です。
- *diff* は 0 です。
- *sec* および *msec* は、両方とも、この時間ウィンドウの、最も古い使用されたメモリ サンプルのタイム スタンプの合計と最新の使用されたメモリ サンプルの合計のタイム スタンプとの間の、実際の時間の差です。

is_percent 引数が TRUE であり、イベント登録 Tcl コマンド拡張で時間ウィンドウがゼロよりも大きい値に指定されている場合は、次のようになります。

- *used* は 0 です。
- *avail* は 0 です。
- *diff* は、指定された時間ウィンドウの、最も古い使用されたメモリ サンプルの合計と最新の使用されたメモリ サンプルの合計との間の、パーセンテージによる差です。
- *sec* および *msec* は、両方とも、この時間ウィンドウの、最も古い使用されたメモリ サンプルのタイム スタンプの合計と最新の使用されたメモリ サンプルのタイム スタンプの合計との間の、実際の時間の差です。

is_percent 引数が TRUE であり、イベント登録 Tcl コマンド拡張で sec 引数と msec 引数が 0 に指定されているか、または指定されていない場合は、次のようになります。

- *used* は 0 です。
- *avail* は 0 です。
- *diff* は、今まで収集された、使用された最初のメモリ サンプルの合計と、プロセスで使用された最新のメモリ サンプルの合計との間の、パーセンテージによる差です。
- *sec* および *msec* は、今まで収集された、使用された最初のメモリ サンプルの合計と使用された最新のメモリ サンプルの合計との間の、実際の時間の差です。

_cerrno を設定

Yes

Embedded Event Manager イベント パブリッシュ Tcl コマンド拡張

event_publish appl

アプリケーション固有のイベントをパブリッシュします。

構文

```
event_publish sub_system ? type ? [arg1 ?] [arg2 ?] [arg3 ?] [arg4 ?]
```

引数

sub_system	（必須）アプリケーション固有のイベントをパブリッシュした EEM ポリシーに割り当てられる番号。 他のすべての番号は Cisco での使用のために予約されているため、番号は 798 に設定されます。
type	（必須）指定されたコンポーネント内のイベント サブタイプ。 sub_system 引数および type 引数によって、アプリケーション イベントが一意に識別されます。 1 ～ 4294967295 の範囲の整数である必要があります。
[arg1 ?]-[arg4 ?]	（任意）4 つのアプリケーション イベントのパブリッシャの文字列データ。

結果文字列

なし

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティング システムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

使用例

この例では、同じ機能（たとえば、Tcl 文の指定されたグループによって CPU 時間の長さを測定する）を実行するため、**event_publish appl** Tcl コマンド拡張を使用して、スクリプトを *n* 回実行する方法を示します。 この例では、2 つの Tcl スクリプトが使用されます。

Script1 によって、タイプ 9999 EEM イベントがパブリッシュされ、Script2 の 1 回目の実行が行われます。 Script1 は、none イベントとして登録され、Cisco IOS XR ソフトウェア CLI **event manager run** コマンドを使用して実行されます。 Script2 は、タイプ 9999 の EEM アプリケーション イベントとして登録され、このスクリプトによって、アプリケーションによってパブリッシュされた arg1 データ（繰り返し回数）が、EEM 環境変数 test_iterations の値を超過したかどうかがチェックされます。 test_iterations の値が超えた場合、スクリプトによってメッセージが書き込まれ、終了します。これ以外の場合、スクリプトによって残りの文が実行され、別の実行が再スケジュールされます。 Script2 の CPU 使用率を測定するには、10 の倍数である test_iterations の値を使用して、Script2 によって使用される CPU 時間の平均の長さを計算します。

Tcl スクリプトを実行するには、次の Cisco IOS XR ソフトウェア コマンドを使用します。

```
configure terminal
 event manager environment test_iterations 100
 event manager policy script1.tcl
 event manager policy script2.tcl
 end
event manager run script1.tcl
```

Tcl スクリプト Script2 によって、100 回実行されます。 余分な処理なしてスクリプトを実行し、CPU 使用率の平均を導き出し、次に余分な処理を追加して、テストを繰り返す場合、以降の CPU 使用率から前の CPU 使用率を差し引き、余分な処理の平均を調べることができます。

Script1（script1.tcl）

```
::cisco::eem::event_register_none

namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

# Query the event info.
array set arr_einfo [event_reqinfo]
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}

action_syslog priority info msg "EEM application_publish test start"
if {$_cerrno != 0} {
    set result [format \
          "component=%s; subsys err=%s; posix err=%s;\n%s" \
          $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}

# Cause the first iteration to run.
event_publish sub_system 798 type 9999 arg1 0
if {$_cerrno != 0} {
    set result [format \
          "component=%s; subsys err=%s; posix err=%s;\n%s" \
          $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
```

Script2（script2.tcl）

```
::cisco::eem::event_register_appl sub_system 798 type 9999

# Check if all the required environment variables exist.
# If any required environment variable does not exist, print out an error msg and quit.
if {![info exists test_iterations]} {
    set result \
      "Policy cannot be run: variable test_iterations has not been set"
    error $result $errorInfo
}

namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

# Query the event info.
array set arr_einfo [event_reqinfo]
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
        $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
# Data1 contains the arg1 value used to publish this event.
```

```
set iter $arr_einfo(data1)

# Use the arg1 info from the previous run to determine when to end.
if {$iter >= $test_iterations} {
    # Log a message.
    action_syslog priority info msg "EEM application_publish test end"
    if {$_cerrno != 0} {
      set result [format \
            "component=%s; subsys err=%s; posix err=%s;\n%s" \
            $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
      error $result
    }
    exit 0
}
set iter [expr $iter + 1]

# Log a message.
set msg [format "EEM application_publish test iteration %s" $iter]
action_syslog priority info msg $msg
if {$_cerrno != 0} {
    set result [format "component=%s; subsys err=%s; posix err=%s;\n%s" \
      $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}

# Do whatever processing that you want to measure here.

# Cause the next iteration to run. Note that the iteration is passed to the
# next operation as arg1.
event_publish sub_system 798 type 9999 arg1 $iter
if {$_cerrno != 0} {
    set result [format \
          "component=%s; subsys err=%s; posix err=%s;\n%s" \
          $_cerr_sub_num $_cerr_sub_err $_cerr_posix_err $_cerr_str]
    error $result
}
```

Embedded Event Manager アクション Tcl コマンド拡張

action_process

ソフトウェア モジュール方式プロセスを起動、再起動、または停止します。 この Tcl コマンド拡張は、ソフトウェア モジュール方式イメージでのみサポートされます。

構文

```
action_process start|restart|kill [job_id ?]
[process_name ?] [instance ?]
```

引数

start	（必須）プロセスが起動されるよう指定します。
restart	（必須）プロセスが再起動されるよう指定します。

kill	（必須）プロセスが停止されるよう指定します。
job_id	（任意）システムマネージャによってプロセスに割り当てられるジョブ ID。 この引数を指定する場合、1 ～ 4294967295 の範囲の整数である必要があります。
process_name	（任意）プロセス名。 job_id を指定するか、または、process_name および instance を指定する必要があります。
instance	（任意）プロセス インスタンス ID。 この引数を指定する場合、1 ～ 4294967295 の範囲の整数である必要があります。

結果文字列

なし

_cerrno を設定

Yes

```
(_cerr_sub_err = 14)    FH_ENOSUCHACTION  (unknown action type)
```

このエラーは、要求されたアクション コマンドが未知であることを示します。

```
(_cerr_sub_num = 425, _cerr_sub_err = 1) SYSMGR_ERROR_INVALID_ARGS  (Invalid arguments
passed)
```

このエラーは、渡された引数が無効であったことを意味します。

```
(_cerr_sub_num = 425, _cerr_sub_err = 2) SYSMGR_ERROR_NO_MEMORY  (Could not allocate required
 memory)
```

このエラーは、メモリの内部 SYSMGR 要求に障害が発生したことを意味します。

```
(_cerr_sub_num = 425, _cerr_sub_err = 5) SYSMGR_ERROR_NO_MATCH  (This process is not known
 to sysmgr)
```

このエラーは、プロセス名が未知であったことを意味します。

```
(_cerr_sub_num = 425, _cerr_sub_err = 14) SYSMGR_ERROR_TOO_BIG  (outside the valid limit)
```

このエラーは、オブジェクト サイズがその最大値を超えたことを意味します。

```
(_cerr_sub_num = 425, _cerr_sub_err = 15) SYSMGR_ERROR_INVALID_OP  (Invalid operation for
this process)
```

このエラーは、その動作がプロセスに対して無効であったことを意味します。

action_program

Tcl スクリプトで、POSIX プロセス（プログラム）を実行できるようにします。任意で、該当する引数文字列、環境文字列、標準入力（stdin）パス名、標準出力（stdout）パス名、または標準エラー（stderr）パス名を使用します。 この Tcl コマンド拡張は、ソフトウェア モジュール方式イメージでのみサポートされます。

構文

```
action_program path ? [argv ?] [envp ?] [stdin ?] [stdout ?] [stderr ?]
```

引数

path	（必須）実行するプログラムのパス名。
argv	（任意）プログラムの引数文字列。
envp	（任意）プログラムの環境文字列。
stdin	（任意）stdin のパス名。
stdout	（任意）stdout のパス名。
stderr	（任意）stderr のパス名。

結果文字列

なし

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティングシステムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err = 14)    FH_ENOSUCHACTION  (unknown action type)
```

このエラーは、要求されたアクション コマンドが未知であることを示します。

```
(_cerr_sub_err = 34)    FH_EMAXLEN  (maximum length exceeded)
```

このエラーは、オブジェクト長またはオブジェクト数が、最大値を超えたことを意味します。

action_script

Tcl スクリプトで、すべての Tcl スクリプトの実行をイネーブルまたはディセーブルにします（スクリプト スケジューラをイネーブルまたはディセーブルにします）。

構文

```
action_script [status enable|disable]
```

引数

status	（任意）スクリプト実行ステータスを示すフラグ。この引数がイネーブルに設定されている場合、スクリプト実行がイネーブルにされます。この引数がディセーブルに設定されている場合、スクリプト実行がディセーブルにされます。

結果文字列

なし

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティング システムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err = 14)    FH_ENOSUCHACTION  (unknown action type)
```

このエラーは、要求されたアクション コマンドが未知であることを示します。

```
(_cerr_sub_err = 52)    FH_ECONFIG  (configuration error)
```

このエラーは、設定エラーが発生したことを意味します。

action_setver_prior

絶対パスで示されるプロセスを以前のバージョンに戻します。

構文

```
action_setver_prior [path ?]
```

引数

path	（必須）プロセスの実行可能パス。

結果文字列

なし

_cerrno を設定

Yes

action_setnode

以降の EEM コマンドをそのノードで実行できるようにするために、指定されたノードに切り替えます。 次の EEM コマンドは action_setnode を使用してそのターゲット ノードを設定します。

- action_process
- sys_reqinfo_proc
- sys_reqinfo_proc_all
- sys_reqinfo_crash_history
- sys_reqinfo_proc_version

構文

```
action_setnode [node ?]
```

引数

node	（必須）ノードの名前。

結果文字列

なし

_cerrno を設定

Yes

action_syslog

メッセージを記録します。

構文

```
action_syslog [priority emerg|alert|crit|err|warning|notice|info|debug]
[msg ?]
```

引数

priority	（任意）action_syslog メッセージ ファシリティ レベル。この引数が指定されない場合、デフォルトのプライオリティは LOG_INFO です。
msg	（任意）記録されるメッセージ。

結果文字列

なし

_cerrno を設定

Yes

```
(_cerr_sub_err = 14)    FH_ENOSUCHACTION  (unknown action type)
```

このエラーは、要求されたアクション コマンドが未知であることを示します。

Embedded Event Manager ユーティリティ Tcl コマンド拡張

appl_read

Embedded Event Manager（EEM）アプリケーションの揮発性データを読み取ります。 この Tcl コマンド拡張では、EEM アプリケーションの揮発性データの読み取りがサポートされます。 EEM アプリケーションの揮発性データは、API をパブリッシュする EEM アプリケーションが使用される Cisco IOS XR ソフトウェア プロセスによってパブリッシュすることができます。 EEM アプリケーションの揮発性データは、EEM ポリシーによってパブリッシュできません。

（注） 現在、アプリケーション揮発性データをパブリッシュする Cisco IOS XR ソフトウェア プロセスはありません。

構文

```
appl_read name ? length ?
```

引数

name	（必須）アプリケーションによってパブリッシュされる文字列データの名前。
length	（必須）読み取る文字列データの長さ。 1 ～ 4294967295 の範囲の整数である必要があります。

結果文字列

```
data %s
```

data は、読み取られる、アプリケーションによってパブリッシュされた文字列データです。

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティングシステムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err =  7)    FH_ENOSUCHKEY  (could not find key)
```

このエラーは、アプリケーション イベント ディテクタ情報キーまたはその他の ID が見つからなかったことを意味します。

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

このエラーは、メモリの内部 EEM 要求に障害が発生したことを意味します。

appl_reqinfo

Embedded Event Manager（EEM）から、前に保存された情報が取得されます。 この Tcl コマンド拡張によって、一意のキーで前に保存された EEM からの情報の取得がサポートされます。これは、情報を取得するために指定する必要があります。 情報の取得によって、その情報が EEM から削除されることに、注意してください。 再度取得できるようにするには、再保存する必要があります。

構文

```
appl_reqinfo key ?
```

引数

key	（必須）データの文字列キー。

結果文字列

```
data %s
```

data は、取得されるアプリケーション文字列データです。

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティングシステムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err =  7)    FH_ENOSUCHKEY  (could not find key)
```

このエラーは、アプリケーション イベント ディテクタ情報キーまたはその他の ID が見つからなかったことを意味します。

appl_setinfo

EEM に情報を保存します。 この Tcl コマンド拡張によって、同じポリシーまたは別のポリシーによって、後で取得できる EEM への情報の保存がサポートされます。 一意のキーを指定する必要があります。 このキーを使用すると、情報を後で取得することができます。

構文

```
appl_setinfo key ? data ?
```

引数

key	（必須）データの文字列キー。
data	（必須）保存するアプリケーション文字列データ。

結果文字列

なし

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティングシステムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err =  8)    FH_EDUPLICATEKEY  (duplicate appl info key)
```

このエラーは、アプリケーション イベント ディテクタ情報キーまたはその他の ID が重複していたことを意味します。

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

このエラーは、メモリの内部 EEM 要求に障害が発生したことを意味します。

```
(_cerr_sub_err = 34)    FH_EMAXLEN  (maximum length exceeded)
```

このエラーは、オブジェクト長またはオブジェクト数が、最大値を超えたことを意味します。

```
(_cerr_sub_err = 43)    FH_EBADLENGTH  (bad API length)
```

このエラーは、API メッセージ長が無効であったことを意味します。

counter_modify

カウンタの値を変更します。

構文

```
counter_modify event_id ? val ? op nop|set|inc|dec
```

引数

event_id	（必須）**register_counter** Tcl コマンド拡張によって返されるカウンタ イベント ID。 0 ～ 4294967295 の範囲の整数である必要があります。

val	（必須） • op が設定されている場合、この引数は、設定されるカウンタ値を表します。 • op が inc の場合、この引数は、カウンタを増やすために使用される値です。 • op が dec の場合、この引数は、カウンタを減らすために使用される値です。
op	（必須） • nop：現在のカウンタの値を取得します。 • set：カウンタの値を指定値に設定します。 • inc：カウンタの値を指定値分増やします。 • dec：カウンタの値を指定値分減らします。

結果文字列

```
val_remain %d
```

val_remain は、カウンタの現在の値です。

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティングシステムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

このエラーは、イベントが登録されたときか、またはイベント ディテクタの内部イベント構造が破損したときに、イベント指定 ID を照会できなかったことを意味します。

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

このエラーは、内部EEMイベントディテクタポインタに値が含まれている必要があったときに、ヌルであったことを意味します。

```
(_cerr_sub_err = 30)    FH_ECTBADOPER  (bad counter threshold operator)
```

このエラーは、カウンタイベントディテクタの設定演算子または変更演算子が、無効であったことを意味します。

fts_get_stamp

最後にソフトウェアがブートされて以来の経過時間を返します。 この Tcl コマンド拡張を使用すると、配列 nsec nnnn に、ブート以降のナノ秒数が返されます。nnnn は、ナノ秒数です。

構文

```
fts_get_stamp
```

引数

なし

結果文字列

```
nsec %d
```
nsec は、ブート以降のナノ秒数です。

_cerrno を設定

No

register_counter

カウンタを登録し、カウンタ イベント ID を返します。 この Tcl コマンド拡張は、カウンタのパブリッシャによって使用され、イベント ID を使用してカウンタを操作する前に、この登録が実行されます。

構文

```
register_counter name ?
```

引数

name	（必須）操作されるカウンタの名前。

結果文字列

```
event_id %d
event_spec_id %d
```
event_id は、指定されたカウンタのカウンタ イベント ID です。**unregister_counter** Tcl コマンド拡張または **counter_modify** Tcl コマンド拡張によって、カウンタの操作に使用されます。 event_spec_id 引数は、指定されたカウンタのイベント指定 ID です。

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティング システムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err =  4)    FH_EINITONCE  (Init() is not yet done, or done twice.)
```

このエラーは、EEM イベント ディテクタがその初期化を完了する前に、特定のイベントを登録する要求が行われたことを意味します。

```
(_cerr_sub_err =  6)    FH_EBADEVENTTYPE  (unknown EEM event type)
```

このエラーは、内部イベント指定で指定されたイベント タイプが無効であったことを意味します。

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

このエラーは、メモリの内部 EEM 要求に障害が発生したことを意味します。

```
(_cerr_sub_err = 10)    FH_ECORRUPT  (internal EEM API context is corrupt)
```

このエラーは、内部 EEM API コンテキスト構造が破損したことを意味します。

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

このエラーは、イベントが登録されたときか、またはイベント ディテクタの内部イベント構造が破損したときに、イベント指定 ID を照会できなかったことを意味します。

```
(_cerr_sub_err = 12)    FH_ENOSUCHEID  (unknown event ID)
```

このエラーは、イベントが登録されたときか、またはイベント ディテクタの内部イベント構造が破損したときに、イベント ID を照会できなかったことを意味します。

```
(_cerr_sub_err = 16)    FH_EBADFMPPTR  (bad ptr to fh_p data structure)
```

このエラーは、各 EEM API コールで使用されるコンテキスト ポインタが不正確であったことを意味します。

```
(_cerr_sub_err = 17)    FH_EBADADDRESS  (bad API control block address)
```

このエラーは、EEM API に渡された制御ブロック アドレスが不正確であったことを意味します。

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

このエラーは、内部 EEM イベント ディテクタ ポインタに値が含まれている必要があったときに、ヌルであったことを意味します。

```
(_cerr_sub_err = 25)    FH_ESUBSEXCEED  (number of subscribers exceeded)
```

このエラーは、タイマーまたはカウンタのサブスクライバの数が、最大値を超えたことを意味します。

```
(_cerr_sub_err = 26)    FH_ESUBSIDXINV  (invalid subscriber index)
```

これは、サブスクライバの索引が無効であったことを意味します。

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

このエラーは、イベント ディテクタが使用できなかったことを意味します。

```
(_cerr_sub_err = 56)    FH_EFDCONNERR  (event detector connection error)
```

このエラーは、この要求を処理するEEMイベントディテクタは使用できないことを意味します。

register_timer

タイマーを登録し、タイマー イベント ID を返します。 この Tcl コマンド拡張は、カウンタのパブリッシャによって使用され、パブリッシャまたはサブスクライバとしての登録に、**event_register_timer** コマンド拡張が使用されなかった場合に、イベントIDを使用してタイマーを操作する前に、この登録が実行されます。

構文

```
register_timer watchdog|countdown|absolute|cron name ?
```

引数

name	（必須）操作されるタイマーの名前。

結果文字列

```
event_id %u
```

event_id は、指定されたタイマーのタイマー イベント ID です（**timer_arm** コマンド拡張または **timer_cancel** コマンド拡張によってタイマーの操作に使用できます）。

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティングシステムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err =  4)    FH_EINITONCE  (Init() is not yet done, or done twice.)
```

このエラーは、EEMイベントディテクタがその初期化を完了する前に、特定のイベントを登録する要求が行われたことを意味します。

```
(_cerr_sub_err =  6)    FH_EBADEVENTTYPE  (unknown EEM event type)
```

このエラーは、内部イベント指定で指定されたイベント タイプが無効であったことを意味します。

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

このエラーは、メモリの内部 EEM 要求に障害が発生したことを意味します。

```
(_cerr_sub_err = 10)    FH_ECORRUPT  (internal EEM API context is corrupt)
```

このエラーは、内部 EEM API コンテキスト構造が破損したことを意味します。

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

このエラーは、イベントが登録されたときか、またはイベント ディテクタの内部イベント構造が破損したときに、イベント指定 ID を照会できなかったことを意味します。

```
(_cerr_sub_err = 16)    FH_EBADFMPPTR  (bad ptr to fh_p data structure)
```

このエラーは、各 EEM API コールで使用されるコンテキスト ポインタが不正確であったことを意味します。

```
(_cerr_sub_err = 17)    FH_EBADADDRESS  (bad API control block address)
```

このエラーは、EEM API に渡された制御ブロック アドレスが不正確であったことを意味します。

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

このエラーは、内部EEMイベントディテクタ ポインタに値が含まれている必要があったときに、ヌルであったことを意味します。

```
(_cerr_sub_err = 25)    FH_ESUBSEXCEED  (number of subscribers exceeded)
```

このエラーは、タイマーまたはカウンタのサブスクライバの数が、最大値を超えたことを意味します。

```
(_cerr_sub_err = 26)    FH_ESUBSIDXINV  (invalid subscriber index)
```

これは、サブスクライバの索引が無効であったことを意味します。

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

このエラーは、イベント ディテクタが使用できなかったことを意味します。

```
(_cerr_sub_err = 56)    FH_EFDCONNERR  (event detector connection error)
```

このエラーは、この要求を処理するEEMイベントディテクタは使用できないことを意味します。

timer_arm

タイマーを搭載します。 タイプは、CRON、ウォッチドッグ、カウントダウン、または絶対の場合があります。

構文

```
timer_arm event_id ? cron_entry ?|time ?
```

引数

event_id	（必須）**register_timer** Tcl コマンド拡張によって返されるタイマー イベント ID。 0 ～ 4294967295 の範囲の整数である必要があります。
cron_entry	（必須）タイマー タイプが CRON の場合に存在する必要があります。他のタイプのタイマーの場合には、存在させることはできません。CRON タイマー指定によって、CRON テーブル エントリの形式が使用されます。
time	（必須）タイマー タイプが CRON ではない場合に存在する必要があります。 タイマー タイプが CRON の場合には、存在できません。ウォッチドッグ タイマーおよびカウントダウン タイマーでは、タイマーの期限が切れるまでの秒数およびミリ秒数です。絶対タイマーでは、期限切れ時刻のカレンダー時間です（SSSSSSSSSS[.MMM] 形式で指定します。SSSSSSSSSS は、0 ～ 4294967295 の秒数を表す整数で、MMM は 0 ～ 999 のミリ秒数を表す整数である必要があります）。 期限の絶対日付は、1970 年 1 月 1 日以降の秒およびミリ秒の単位での数です。 指定された日付がすでに過ぎた場合、タイマーの期限はただちに切れます。

結果文字列

```
sec_remain %ld msec_remain %ld
```

sec_remain および msec_remain は、タイマーの次の期限切れまでの残り時間です。

（注） タイマー タイプが CRON の場合、sec_remain 引数および msec_remain 引数には 0 が返されます。

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティング システムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err =  6)    FH_EBADEVENTTYPE  (unknown EEM event type)
```

このエラーは、内部イベント指定で指定されたイベント タイプが無効であったことを意味します。

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

このエラーは、メモリの内部 EEM 要求に障害が発生したことを意味します。

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

このエラーは、イベントが登録されたときか、またはイベント ディテクタの内部イベント構造が破損したときに、イベント指定 ID を照会できなかったことを意味します。

```
(_cerr_sub_err = 12)    FH_ENOSUCHEID  (unknown event ID)
```

このエラーは、イベントが登録されたときか、またはイベント ディテクタの内部イベント構造が破損したときに、イベント ID を照会できなかったことを意味します。

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

このエラーは、内部EEMイベント ディテクタ ポインタに値が含まれている必要があったときに、ヌルであったことを意味します。

```
(_cerr_sub_err = 27)    FH_ETMDELAYZR  (zero delay time)
```

このエラーは、タイマーの搭載に指定された時間がゼロであったことを意味します。

```
(_cerr_sub_err = 42)    FH_ENOTREGISTERED  (request for event spec that is unregistered)
```

このエラーは、イベント検出が登録できなかったことを意味します。

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

このエラーは、イベント ディテクタが使用できなかったことを意味します。

```
(_cerr_sub_err = 56)    FH_EFDCONNERR  (event detector connection error)
```

このエラーは、この要求を処理するEEMイベントディテクタは使用できないことを意味します。

timer_cancel

タイマーを取り消します。

構文

```
timer_cancel event_id ?
```

引数

event_id	（必須）**register_timer** Tcl コマンド拡張によって返されるタイマー イベント ID。 0 ～ 4294967295 の範囲の整数である必要があります。

結果文字列

```
sec_remain %ld msec_remain %ld
```

sec_remain および msec_remain は、タイマーの次の期限切れまでの残り時間です。

（注） タイマー タイプが CRON の場合、sec_remain および msec_remain には 0 が返されます。

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティングシステムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err =  6)    FH_EBADEVENTTYPE  (unknown EEM event type)
```

このエラーは、内部イベント指定で指定されたイベント タイプが無効であったことを意味します。

```
(_cerr_sub_err =  7)    FH_ENOSUCHKEY  (could not find key)
```

このエラーは、アプリケーション イベント ディテクタ情報キーまたはその他の ID が見つからなかったことを意味します。

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

このエラーは、イベントが登録されたときか、またはイベント ディテクタの内部イベント構造が破損したときに、イベント指定 ID を照会できなかったことを意味します。

```
(_cerr_sub_err = 12)    FH_ENOSUCHEID  (unknown event ID)
```

このエラーは、イベントが登録されたときか、またはイベント ディテクタの内部イベント構造が破損したときに、イベント ID を照会できなかったことを意味します。

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

このエラーは、内部EEMイベントディテクタポインタに値が含まれている必要があったときに、ヌルであったことを意味します。

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

このエラーは、イベント ディテクタが使用できなかったことを意味します。

```
(_cerr_sub_err = 56)    FH_EFDCONNERR  (event detector connection error)
```

このエラーは、この要求を処理するEEMイベントディテクタは使用できないことを意味します。

unregister_counter

カウンタの登録を解除します。 この Tcl コマンド拡張は、カウンタのパブリッシャによって使用され、**register_counter** コマンド拡張で前に登録されたカウンタの登録を解除します。

構文

```
unregister_counter event_id ? event_spec_id ?
```

引数

event_id	（必須）**register_counter** コマンド拡張によって返されるカウンタ イベント ID。 0 ～ 4294967295 の範囲の整数である必要があります。
event_spec_id	（必須）**register_counter** コマンド拡張によって返された、指定されたカウンタのカウンタ イベント指定 ID。 0 ～ 4294967295 の範囲の整数である必要があります。

結果文字列

なし

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティングシステムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

このエラーは、メモリの内部 EEM 要求に障害が発生したことを意味します。

```
(_cerr_sub_err = 11)    FH_ENOSUCHESID  (unknown event specification ID)
```

このエラーは、イベントが登録されたときか、またはイベント ディテクタの内部イベント構造が破損したときに、イベント指定 ID を照会できなかったことを意味します。

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

このエラーは、内部EEMイベントディテクタ ポインタに値が含まれている必要があったときに、ヌルであったことを意味します。

```
(_cerr_sub_err = 26)    FH_ESUBSIDXINV  (invalid subscriber index)
```

これは、サブスクライバの索引が無効であったことを意味します。

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

このエラーは、イベント ディテクタが使用できなかったことを意味します。

```
(_cerr_sub_err = 56)    FH_EFDCONNERR  (event detector connection error)
```

このエラーは、この要求を処理するEEMイベントディテクタは使用できないことを意味します。

Embedded Event Manager システム情報 Tcl コマンド拡張

（注） すべての EEM システム情報コマンド **sys_reqinfo _xxx** には、**yes** に設定された「_cerrno を設定」セクションがあります。

sys_reqinfo_cpu_all

指定された期間で、指定された順序で、上位プロセスのCPU使用率（POSIXプロセスとIOSプロセスの両方）を問い合わせます。 この Tcl コマンド拡張は、ソフトウェア モジュール方式イメージでのみサポートされます。

構文

```
sys_reqinfo_cpu_all order cpu_used [sec ?] [msec ?] [num ?]
```

引数

order	（必須）プロセスの CPU 使用率のソートに使用される順序。
cpu_used	（必須）指定されたウィンドウの、CPU使用率の平均が、降順でソートされるよう、指定します。
secmsec	（任意）CPU使用率の平均が計算される、秒単位およびミリ秒単位での時間。 0 から 4294967295 の範囲の整数である必要があります。 指定されない場合か、または、sec と msec の両方が 0 と指定される場合、最新の CPU サンプルが使用されます。
num	（任意）表示される、ソートされたプロセスのリストの上位からのエントリの数。 1 ～ 4294967295 の範囲の整数である必要があります。 デフォルト値は 5 です。

結果文字列

```
rec_list {{process CPU info string 0},{process CPU info string 1}, ...}
```

各プロセスの CPU 情報文字列は、次のとおりです。

```
pid %u name {%s} cpu_used %u
```

rec_list	プロセス CPU 情報リストの開始をマークします。
pid	プロセス ID。
name	プロセス名。

cpu_used	sec と msec が、ゼロよりも大きい数で指定される場合、平均パーセンテージは、指定された時間のプロセスCPU使用率から計算されるよう、指定します。 sec と msec が、両方ともゼロか、または指定されない場合。平均パーセンテージは、最新のサンプルのプロセス CPU 使用率から計算されます。

_cerrno を設定

Yes

sys_reqinfo_crash_history

クラッシュしたすべてのプロセスのプロセス情報を問い合わせます。 この Tcl コマンド拡張は、ソフトウェア モジュール方式イメージでのみサポートされます。

構文

```
sys_reqinfo_crash_history
```

引数

なし

結果文字列

```
rec_list {{crash info string 0},{crash info string 1}, ...}

Where each crash info string is:

job_id %u name {%s} respawn_count %u fail_count %u dump_count %u
inst_id %d exit_status 0x%x exit_type %d proc_state {%s} component_id 0x%x
crash_time_sec %ld crash_time_msec %ld
```

job_id	システムマネージャによってプロセスに割り当てられるジョブ ID。 1 ～ 4294967295 の整数。
name	プロセス名。
respawn_count	プロセスの再起動の合計回数。
fail_count	プロセスの再起動試行の回数。プロセスが正常に再起動されると、このカウントはゼロにリセットされます。
dump_count	実行されたコア ダンプの回数。

inst_id	プロセス インスタンス ID。
exit_status	プロセスの最後の終了ステータス。
exit_type	最後の終了タイプ。
proc_state	Sysmgr プロセスの状態。 error、forced_stop、hold、init、ready_to_run、run、run_rnode、stop、waitEOltimer、wait_rnode、wait_spawntimer、wait_tpl の 1 つです。
component_id	プロセスが属するコンポーネントのコンポーネントIDに割り当てられているバージョンマネージャ。
crash_time_sec crash_time_msec	1970 年 1 月 1 日以降の秒およびミリ秒の単位で、プロセスがクラッシュした最後の時刻を表します。

_cerrno を設定

Yes

sys_reqinfo_mem_all

指定された期間で、指定された順序で、上位プロセスのメモリの使用状況（POSIX と IOS の両方）を問い合わせます。 この Tcl コマンド拡張は、ソフトウェア モジュール方式イメージでのみサポートされます。

構文

```
sys_reqinfo_mem_all order allocates|increase|used [sec ?] [msec ?] [num ?]
```

引数

order	（必須）プロセスのメモリの使用状況のソートに使用される順序。
allocates	（必須）指定された時間ウィンドウの期間に、メモリの使用状況が、プロセス割り当ての数によって降順でソートされるよう、指定します。

increase	（必須）指定された時間ウィンドウの期間に、メモリの使用状況が、プロセスで増えたメモリのパーセンテージによって降順でソートされるよう、指定します。
used	（必須）メモリが、プロセスによって使用される現在のメモリによってソートされるよう、指定します。
secmsec	（任意）プロセスでのメモリの使用状況が計算される、秒単位およびミリ秒単位での時間。 0 から 4294967295 の範囲の整数である必要があります。 sec と msec の両方が指定され、ゼロではない場合、割り当て数は、該当する時間で収集された最も古いサンプルと最新のサンプルでの、割り当て数の差です。パーセンテージは、該当する時間で収集された最も古いサンプルと最新のサンプルとの、パーセンテージの差分として計算されます。指定されない場合か、または、sec と msec の両方が 0 と指定される場合、収集された最初のサンプルが、最も古いサンプルとして使用されます。つまり、時間は、起動から現在時までの時間で設定されます。
num	（任意）表示される、ソートされたプロセスのリストの上位からのエントリの数。 1 ～ 4294967295 の範囲の整数である必要があります。 デフォルト値は 5 です。

結果文字列

```
rec_list {{process mem info string 0},{process mem info string 1}, ...}
```

各プロセスのメモリ情報文字列は、次のとおりです。

```
pid %u name {%s} delta_allocs %d initial_alloc %u current_alloc %u percent_increase %d
```

rec_list	プロセスのメモリの使用状況情報リストの開始をマークします。
pid	プロセス ID。
name	プロセス名。

delta_allocs	該当する期間で収集された、最も古いサンプルと最新のサンプルでの、割り当て数の差として、差を指定します。
initial_alloc	時間の開始時にプロセスによって使用される、キロバイト単位での、メモリの容量を指定します。
current_alloc	プロセスによって使用される、キロバイト単位での、メモリの容量を指定します。
percent_increase	該当する時間で収集された最も古いサンプルと最新のサンプルとの、使用メモリのパーセンテージの差分を指定します。パーセンテージの差は、current_alloc から initial_alloc の 100 を差し引いた数として、および、initial_alloc で割った数として、表すことができます。

_cerrno を設定

Yes

sys_reqinfo_proc

1 つの POSIX プロセスに関する情報を問い合わせます。 この Tcl コマンド拡張は、ソフトウェアモジュール方式イメージでのみサポートされます。

構文

```
sys_reqinfo_proc job_id ?
```

引数

job_id	（必須）システムマネージャによってプロセスに割り当てられるジョブ ID。 1 ～ 4294967295 の範囲の整数である必要があります。

結果文字列

```
job_id %u component_id 0x%x name {%s} helper_name {%s} helper_path {%s} path {%s}
node_name {%s} is_respawn %u is_mandatory %u is_hold %u dump_option %d
max_dump_count %u respawn_count %u fail_count %u dump_count %u
last_respawn_sec %ld last_respawn_msec %ld inst_id %u proc_state %s
level %d exit_status 0x%x exit_type %d
```

job_id	システムマネージャによってプロセスに割り当てられるジョブ ID。 1 ～ 4294967295 の整数。
component_id	プロセスが属するコンポーネントのコンポーネントIDに割り当てられているバージョンマネージャ。
name	プロセス名。
helper_name	ヘルパー プロセスの名前。
helper_path	ヘルパー プロセスの実行可能パス。
path	プロセスの実行可能パス。
node_name	プロセスが属するノードのノード名に割り当てられているシステム マネージャ。
is_respawn	プロセスが再生成できることを指定するフラグ。
is_mandatory	プロセスが実行され続ける必要があることを指定するフラグ。
is_hold	API によって呼び出されるまでプロセスが再生成されることを指定するフラグ。
dump_option	コア ダンプのオプション。
max_dump_count	許可されるコア ダンプの最大数。
respawn_count	プロセスの再起動の合計回数。
fail_count	プロセスの再起動試行の回数。プロセスが正常に再起動されると、このカウントはゼロにリセットされます。
dump_count	実行されたコア ダンプの回数。
last_respawn_seclast_respawn_msec	1970 年 1 月 1 日以降の POSIX タイマー ユニットでの秒およびミリ秒の単位で、プロセスが開始された最後の時刻を表します。
inst_id	プロセス インスタンス ID。

proc_state	Sysmgr プロセスの状態。 error、forced_stop、hold、init、ready_to_run、run、run_rnode、stop、waitEOltimer、wait_rnode、wait_spawntimer、wait_tpl の 1 つです。
level	プロセス実行レベル。
exit_status	プロセスの最後の終了ステータス。
exit_type	最後の終了タイプ。

_cerrno を設定

Yes

sys_reqinfo_proc_all

すべての POSIX プロセスの情報を問い合わせます。 この Tcl コマンド拡張は、ソフトウェア モジュール方式イメージでのみサポートされます。

構文

```
sys_reqinfo_proc_all
```

引数

なし

結果文字列

```
rec_list {{process info string 0}, {process info string 1},...}
```
各プロセスの情報文字列は、**sysreq_info_proc** Tcl コマンド拡張の結果文字列と同じです。

_cerrno を設定

Yes

sys_reqinfo_proc_version

指定したプロセスのバージョンを問い合わせます。

構文

```
sys_reqinfo_proc_version [job_id ?]
```

引数

job_id	（必須）システムマネージャによってプロセスに割り当てられるジョブ ID。 1 ～ 2147483647 の範囲の整数であることが必要です。

結果文字列

```
version_id %02d.%02d.%04d
```

version_id は、プロセスのバージョン番号が割り当てられたバージョン マネージャです。

_cerrno を設定

Yes

sys_reqinfo_routername

ルータ名を問い合わせます。

構文

```
sys_reqinfo_routername
```

引数

なし

結果文字列

```
routername %s
```

routername は、ルータの名前です。

_cerrno を設定

Yes

sys_reqinfo_syslog_freq

すべての Syslog イベントの頻度情報を問い合わせます。

構文

```
sys_reqinfo_syslog_freq
```

引数

なし

結果文字列

```
rec_list {{event frequency string 0}, {log freq str 1}, ...}
```

各イベントの頻度の文字列は、次のとおりです。

```
time_sec %ld time_msec %ld match_count %u raise_count %u occurs %u
period_sec %ld period_msec %ld pattern {%s}
```

time_sectime_msec	1970 年 1 月 1 日以降の POSIX タイマー ユニットでの秒およびミリ秒の単位で、最後のイベントが発生した時刻を表します。
match_count	イベントの登録以降、この Syslog イベント指定によって指定されたパターンが、Syslog メッセージによって照会される回数。
raise_count	この Syslog イベントが発生した回数。
occurs	イベントを発生させるために必要な発生回数。指定されない場合、イベントは 1 回目から発生します。
period_secperiod_msec	イベントを発生させるには、発生回数が POSIX タイマーユニットのこの数以内である必要があります。この引数が指定されない場合、時間のチェックは適用されません。
pattern	Syslog メッセージのパターン マッチの実行に使用される正規表現。

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティング システムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err =  9)    FH_EMEMORY  (insufficient memory for request)
```

このエラーは、メモリの内部 EEM 要求に障害が発生したことを意味します。

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

このエラーは、内部EEMイベントディテクタポインタに値が含まれている必要があったときに、ヌルであったことを意味します。

```
(_cerr_sub_err = 45)    FH_ESEQNUM  (sequence or workset number out of sync)
```

このエラーは、イベントディテクタシーケンスまたは作業セット番号が無効であったことを意味します。

```
(_cerr_sub_err = 46)    FH_EREGEMPTY  (registration list is empty)
```

このエラーは、イベントディテクタ登録リストが空であったことを意味します。

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

このエラーは、イベントディテクタが使用できなかったことを意味します。

sys_reqinfo_syslog_history

指定された Syslog メッセージの履歴を問い合わせます。

構文

```
sys_reqinfo_syslog_history
```

引数

なし

結果文字列

```
rec_list {{log hist string 0}, {log hist str 1}, ...}
```

各記録の履歴の文字列は、次のとおりです。

```
time_sec %ld time_msec %ld msg {%s}
```

time_sec time_msec	1970 年 1 月 1 日以降の秒およびミリ秒の単位で、メッセージが記録された時刻を表します。
msg	Syslog メッセージ。

_cerrno を設定

Yes

```
(_cerr_sub_err =  2)    FH_ESYSERR  (generic/unknown error from OS/system)
```

このエラーは、オペレーティングシステムによってレポートされたエラーを意味します。エラーとともにレポートされる POSIX errno 値を使用して、オペレーティング システム エラーの原因を調べます。

```
(_cerr_sub_err = 22)    FH_ENULLPTR  (event detector internal error - ptr is null)
```

このエラーは、内部EEMイベントディテクタポインタに値が含まれている必要があったときに、ヌルであったことを意味します。

```
(_cerr_sub_err = 44)    FH_EHISTEMPTY  (history list is empty)
```

このエラーは、履歴のリストが空であったことを意味します。

```
(_cerr_sub_err = 45)    FH_ESEQNUM  (sequence or workset number out of sync)
```

このエラーは、イベントディテクタシーケンスまたは作業セット番号が無効であったことを意味します。

```
(_cerr_sub_err = 54)    FH_EFDUNAVAIL  (connection to event detector unavailable)
```

このエラーは、イベント ディテクタが使用できなかったことを意味します。

sys_reqinfo_stat

名前で指定された統計エントリの値と、必要に応じて第 1 修飾子および第 2 修飾子を問い合わせます。

構文

```
sys_reqinfo_stat [name ?][mod1 ?][mod2 ?]
```

引数

name	（必須）統計データ要素名。
mod_1	（任意）統計データ要素修飾子 1。
mod_2	（任意）統計データ要素修飾子 2。

結果文字列

```
name %s value %s
```

name	統計データ要素名。
value	統計データ要素の値文字列。

_cerrno を設定

Yes

SMTP ライブラリのコマンド拡張

すべてのシンプル メール転送プロトコル（SMTP）ライブラリ コマンドは、::cisco::lib名前空間に属します。

このライブラリを使用するには、ユーザは、電子メールテンプレートファイルを用意する必要があります。 電子メール サービスと電子メール テキストを、**event manager environment** Cisco IOS XR ソフトウェア コマンドライン インターフェイス（CLI）コンフィギュレーション コマンドを使用して設定できるよう、テンプレート ファイルに Tcl グローバル変数を含めることができます。 電子メール テンプレート ファイルでグローバル変数を置き換え、設定された電子メール サーバを使用して、設定された To アドレス、CC アドレス、From アドレス、および Subject 行プロパティで必要な電子メール コンテキストを送信するには、このライブラリにあるコマンドを使用します。

電子メール テンプレート

電子メール テンプレート ファイルの形式は、次のとおりです。

```
Mailservername:<space><the list of candidate SMTP server addresses>
From:<space><the e-mail address of sender>
To:<space><the list of e-mail addresses of recipients>
Cc:<space><the list of e-mail addresses that the e-mail will be copied to>
Subject:<subject line>
<a blank line>
<body>
```

（注） テンプレートには、通常、設定される Tcl グローバル変数が含まれています。

次に、サンプル電子メール テンプレート ファイルを示します。

```
Mailservername: $_email_server
From: $_email_from
To: $_email_to
Cc: $_email_cc
Subject: From router $routername: Process terminated

process name: $process_name
subsystem: $sub_system
exit status: $exit_status
respawn count: $respawn_count
```

エクスポートされる Tcl コマンド拡張

smtp_send_email

電子メールテンプレートファイルのテキストが、すべてのグローバル変数ですでに置き換えられている場合、シンプル メール転送プロトコル（SMTP）を使用して電子メールを送信します。 電

子メール テンプレートによって、候補メール サーバのアドレス、To アドレス、CC アドレス、From アドレス、件名の行、および電子メールの本文が指定されます。

（注） ライブラリが、リストにあるサーバの 1 つに接続できるまで、サーバへの接続が、1 つ 1 つ試行されるよう、候補電子メール サーバのリストを用意できます。

構文

```
smtp_send_email text
```

引数

text	（必須）すべてのグローバル変数ですでに置き換えられた、電子メール テンプレート ファイルのテキスト。

結果文字列

なし

_cerrno を設定

- 1 行目の形式が間違っている：Mailservername：サーバ名のリスト。
- 2 行目の形式が間違っている：From：送信元アドレス。
- 3 行目の形式が間違っている：To：送信先アドレスのリスト。
- 4 行目の形式が間違っている：CC：コピー送信先アドレスのリスト。
- メール サーバへの接続エラー：リモート サーバによって $sock が閉じられている（$sock はメール サーバに開かれているソケットの名前）。
- メール サーバへの接続エラー：$sock 応答コードが service ready greeting ではなく $k である（$sock はメール サーバに開かれているソケットの名前、$k は $sock の応答コード）。
- メール サーバへの接続エラー：すべてのメール サーバ候補に接続できない。
- メール サーバからの接続解除エラー：リモート サーバによって $sock が閉じられている（$sock はメール サーバに開かれているソケットの名前）。

サンプル スクリプト

電子メール テンプレートですべての必要なグローバル変数が定義された後には、次のようになります。

```
if [catch {smtp_subst [file join $tcl_library email_template_sm]} result] {
    puts stderr $result
```

```
    exit 1
}
if [catch {smtp_send_email $result} result] {
    puts stderr $result
    exit 1
}
```

smtp_subst

電子メール テンプレート ファイル e-mail_template の場合、ファイルにある各グローバル変数を、そのユーザ定義値によって置き換えます。 置換後に、ファイルのテキストを返します。

構文

```
smtp_subst e-mail_template
```

引数

e-mail_template	（必須）グローバル変数が、ユーザ定義値によって置き換えられる必要がある、電子メール テンプレート ファイルの名前。 ファイル名の例は /disk0://example.template で、スロット 0 の ATA フラッシュ ディスクの上位レベル ディレクトリにある example.template という名前のファイルを表します。

結果文字列

すべてのグローバル変数で置き換えられた、電子メール テンプレート ファイルのテキスト。

_cerrno を設定

- 電子メール テンプレート ファイルを開けられない。
- 電子メール テンプレート ファイルを閉じられない。

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、ユーザに対し、CLI コマンドを実行し、Tcl 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ネルをオープンし、コマンドを記述してチャ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、ルータによって、異なるユーザ オプションが質問される「Q&A」フェーズがあり、ユーザは、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 スクリプトで異なる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、これは、CLI ライブラリを使用した非対話式コマンドの実行, （156 ページ） および CLI ライブラリを使用した対話式コマンドの実行, （156 ページ） で説明します。

エクスポートされる Tcl コマンド拡張

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

チャネルをクローズできない。

cli_exec

指定されたチャ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。

_cerrno を設定

チャ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
```

_cerrno 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 をチャネル ハンドラに接続します。 チャネル ハンドラを含む配列を返します。

（注） **cli_open** を呼び出すたびに Cisco IOS XR ソフトウェア EXEC セッションが開始され、Cisco IOS XR ソフトウェア vty が割り当てられます。 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 は、**line vty vty-pool** 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 のプールから割り当てられます。 使用可能な vty が 2 つ以下の場合、cli_open ルーチンは失敗し、残りの vty は Telnet で使用できるよう保存され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。

_cerrno 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ルータ 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 チャネル ハンドラ。

結果文字列

読み取られたすべての内容。

_cerrno を設定

ルータ名を取得できない。

（注） この Tcl コマンド拡張によって、ルータ 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 チャネル ハンドラ。

結果文字列

読み取られたすべての内容。

_cerrno 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 チャネル ハンドラ。

結果文字列

読み取られた回線。

_cerrno 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 XR ソフトウェア 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 チャネル ハンドラ。
ptn	（必須）チャネルからコマンド出力を読み取るときに、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。

_cerrno 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_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

_cerrno 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 cli1 $result
}
if [catch {cli_exec $cli1(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、出力とルータ 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 "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、コマンドの出力とルータ 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。Tcl 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
}

# 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、ルータ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 ライブラリでの融通性を描くために示します。

```
# 1. execute the reload command
if [catch {cli_open} result] {
    error $result $errorInfo
} else {
    array set cli1 $result
}
if [catch {cli_write $cli1(fd) "reload"} result] {
    error $result $errorInfo
} else {
    set cmd_output $result
```

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

Tcl コンテキスト ライブラリ コマンド拡張

すべての Tcl コンテキスト ライブラリ コマンド拡張は、::cisco::eem 名前空間に属します。

エクスポートされるコマンド

context_retrieve

該当するコンテキスト名、使用されている可能性があるスカラ変数名、配列型変数名、および配列の索引によって指定される Tcl 変数を取得します。 取得される情報は、自動的に削除されます。

（注） 保存される情報が一度取得されると、自動的に削除されます。 その情報が別のポリシーで必要な場合、（**context_retrieve** コマンド拡張を使用して）それを取得するポリシーも、（**context_save** コマンド拡張を使用して）再度保存する必要があります。

構文

```
context_retrieve ctxt [var] [index_if_array]
```

引数

ctxt	（必須）コンテキスト名。

var	（任意）スカラ変数名または配列型変数名。この引数が指定されない場合、ヌル文字列を定義します。
index_if_array	（任意）配列インデックス。

（注） *var* 引数がスカラ変数の場合、*index_if_array* 引数は無視されます。

var が未指定の場合、コンテキストに保存されている変数テーブル全体を取得します。

var が指定され、*index_if_array* が指定されない場合、または、*index_if_array* が指定されるが *var* がスカラ変数の場合、*var* の値を取得します。

var が指定され、*index_if_array* が指定され、*var* が配列変数の場合、指定された配列エレメントの値を取得します。

結果文字列

保存が実行されたときの状態に、Tcl グローバル変数をリセットします。

_cerrno を設定

- appl_reqinfo エラーが原因で、_cerrno、_cerr_sub_num、_cerr_sub_err、_cerr_posix_err、_cerr_str を表示する文字列。
- 変数がコンテキストにない。

使用例

次に、**context_save** コマンド拡張機能および **context_retrieve** コマンド拡張機能を使用して、データを保存し、取得する例を示します。 例は、保存と取得のペアで示されます。

例 1：保存

var が未指定か、またはパターンが指定される場合、複数の変数をコンテキストに保存します。

```
::cisco::eem::event_register_none

namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

set testvara 123
set testvarb 345
set testvarc 789
if {[catch {context_save TESTCTX "testvar*"} errmsg]} {
      action_syslog msg "context_save failed: $errmsg"
} else {
      action_syslog msg "context_save succeeded"
}
```

例 1：取得

var が未指定の場合、複数の変数をコンテキストから取得します。

```
::cisco::eem::event_register_none

namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

if {[catch {foreach {var value} [context_retrieve TESTCTX] {set $var $value}} errmsg]} {
      action_syslog msg "context_retrieve failed: $errmsg"
} else {
      action_syslog msg "context_retrieve succeeded"
}
if {[info exists testvara]} {
      action_syslog msg "testvara exists and is $testvara"
} else {
      action_syslog msg "testvara does not exist"
}
if {[info exists testvarb]} {
      action_syslog msg "testvarb exists and is $testvarb"
} else {
      action_syslog msg "testvarb does not exist"
}
if {[info exists testvarc]} {
      action_syslog msg "testvarc exists and is $testvarc"
} else {
      action_syslog msg "testvarc does not exist"
}
```

例 2：保存

var が指定される場合、var の値を保存します。

```
::cisco::eem::event_register_none

namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

set testvar 123
if {[catch {context_save TESTCTX testvar} errmsg]} {
      action_syslog msg "context_save failed: $errmsg"
} else {
      action_syslog msg "context_save succeeded"
}
```

例 2：取得

var が指定され、index_if_array が指定されない場合、または、index_if_array が指定されるが var がスカラ変数の場合、var の値を取得します。

```
::cisco::eem::event_register_none

namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

if {[catch {set testvar [context_retrieve TESTCTX testvar]} errmsg]} {
      action_syslog msg "context_retrieve failed: $errmsg"
} else {
      action_syslog msg "context_retrieve succeeded"
}
if {[info exists testvar]} {
      action_syslog msg "testvar exists and is $testvar"
} else {
      action_syslog msg "testvar does not exist"
```

```
}
```

例 3：保存

var が指定される場合、それが配列の場合でも、var の値を保存します。

```
::cisco::eem::event_register_none

namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

array set testvar "testvar1 ok testvar2 not_ok"
if {[catch {context_save TESTCTX testvar} errmsg]} {
      action_syslog msg "context_save failed: $errmsg"
} else {
      action_syslog msg "context_save succeeded"
}
```

例 3：取得

var が指定され、index_if_array が指定されず、var が配列変数の場合、配列全体を取得します。

```
::cisco::eem::event_register_none

namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

if {[catch {array set testvar [context_retrieve TESTCTX testvar]} errmsg]} {
      action_syslog msg "context_retrieve failed: $errmsg"
} else {
      action_syslog msg "context_retrieve succeeded"
}
if {[info exists testvar]} {
      action_syslog msg "testvar exists and is [array get testvar]"
} else {
      action_syslog msg "testvar does not exist"
}
```

例 4：保存

var が指定される場合、それが配列の場合でも、var の値を保存します。

```
::cisco::eem::event_register_none

namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

array set testvar "testvar1 ok testvar2 not_ok"
if {[catch {context_save TESTCTX testvar} errmsg]} {
      action_syslog msg "context_save failed: $errmsg"
} else {
      action_syslog msg "context_save succeeded"
}
```

例 4：取得

var が指定され、index_if_array が指定され、var が配列変数の場合、指定された配列エレメントの値を取得します。

```
::cisco::eem::event_register_none
```

```
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*

if {[catch {set testvar [context_retrieve TESTCTX testvar testvar1]} errmsg]} {
      action_syslog msg "context_retrieve failed: $errmsg"
} else {
      action_syslog msg "context_retrieve succeeded"
}
if {[info exists testvar]} {
      action_syslog msg "testvar exists and is $testvar"
} else {
      action_syslog msg "testvar doesn't exist"
}
```

context_save

現在およびグローバルな名前空間で、指定されたパターンが、識別情報として指定されたコンテキスト名と一致する、Tcl 変数を保存します。 この Tcl コマンド拡張を使用すると、ポリシー外の情報が保存されます。 保存された情報は、**context_retrieve** コマンド拡張を使用して、異なるポリシーによって取得できます。

（注） 保存される情報が一度取得されると、自動的に削除されます。 その情報が別のポリシーで必要な場合、（**context_retrieve** コマンド拡張を使用して）それを取得するポリシーも、（**context_save** コマンド拡張を使用して）再度保存する必要があります。

構文

```
context_save ctxt [pattern]
```

引数

ctxt	（必須）コンテキスト名。
pattern	（任意）**string match** Tcl コマンドによって使用される、glob-style パターン。 この引数が指定されない場合、パターンのデフォルトは、ワイルドカード * です。 glob パターンで使用されている、3 つの構成があります。 • * = すべての文字 • ? = 1 文字 • [abc] = 文字のセットの 1 つと照合

結果文字列

なし

_cerrno を設定

appl_setinfo エラーが原因で、_cerrno、_cerr_sub_num、_cerr_sub_err、_cerr_posix_err、_cerr_str を表示する文字列。

使用例

context_save コマンド拡張機能および **context_retrieve** コマンド拡張機能を使用して、データの保存や取得を行う方法の例については、使用例, （159 ページ）を参照してください。