EEM イベントの Tcl コマンド拡張

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。

priority low|normal|high

（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。

event_completion

トリガーしたイベントのサービスが行われている EEM サーバーに、通知を送信します。イベントでは、このイベント インスタンスの **return_code** である 1 つの引数のみが使用されます。

構文

```
event_completion status ?
```

引数

status	（必須）このイベントインスタンスの終了ステータス（return_code）。ゼロの値によって、エラーがないことが示され、他のすべての整数によって、エラーが示されます。

結果文字列

なし

_cerrno を設定

なし

event_completion_with_wait

event_completion_with_wait コマンドは、2 つのコマンド、**event_completion** と **event_wait** を使いやすいように 1 つのコマンドに組み合わせたものです。

event_completion コマンドによって、ポリシーをトリガーしたイベントに対してポリシーがサービスを実行したことが EEM サーバーに通知されます。イベントでは、このイベント インスタンスの **return_code** である 1 つの引数のみが使用されます。

event_wait ポリシーがスリープ状態になります。Tcl ポリシーで、新しいイベントを通知する新しい信号を受信すると、ポリシーは使用状態になり、再度スリープ状態に戻ります。このループが継続されます。**event_wait** ポリシーは、**event_completed** ポリシーの前に起動され、エラーが発生して、ポリシーが終了します。

構文

```
event_completion_with_wait status ? [refresh_vars]
```

引数

status	（必須）このイベント インスタンスの exit_status（return_code）。ゼロの値は、エラーがないことを示します。他のすべての整数は、エラーを示します。
refresh_vars	（任意）組み込み変数と環境変数は、このイベントインスタンス中に EEM Policy Director からアップデート（リフレッシュ）する必要があるかどうかを示します。

結果文字列

なし

_cerrno を設定

可

使用例

この 1 つのコマンドを使用した前述の例の類似例を示します。

```
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
set i 1
while {1 == 1} {  # Start high performance policy loop
   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 msg "event $i serviced" priority info
   if {$i == 5} {
       action_syslog msg "Exiting after servicing 5 events" priority info
       exit 0
   }
   incr i
   array set _event_state_arr [event_completion_with_wait status 0 refresh_vars 1]
   if {$_event_state_arr(event_state) != 0} {
       action_syslog msg "Exiting: failed event_state " \
                         " $event_state_arr(event_state)" priority info
       exit 0
   }
}
```

（注） 実行される設定の出力は、event_publish Tcl コマンドと同じです。

event_publish

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

可

```
(_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

次に、ある機能（Tcl ステートメントの所定のグループによって CPU 時間の長さを測定するなど）を実行するため、**event_publish** Tcl コマンド拡張を使用してスクリプトを *n* 回、反復して実行する例を示します。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 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 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
}
```

event_wait

Tclポリシーがスリープ状態になります。Tclポリシーで、新しいイベントを通知する新しい信号を受信すると、ポリシーは使用状態になり、再度スリープ状態に戻ります。このループが継続されます。**event_wait** ポリシーは、**event_completed** ポリシーの前に起動され、エラーが発生して、ポリシーが終了します。

構文

```
event_wait [refresh_vars]
```

引数

refresh_vars	（任意）組み込み変数と環境変数は、このイベントインスタンス中にEEM Policy Directorからアップデート（リフレッシュ）する必要があるかどうかを示します。

結果文字列

なし

_cerrno を設定

なし

使用例

event_wait イベント ディテクタは、**event_state** という名前の単一要素でアレイ タイプ値を返します。Event_state は、イベントの処理中にエラーが発生したかどうかを示す EEM サーバー

から戻される値です。この場合のエラーの例は、ユーザーがイベントインスタンスを処理するときに、**event_completion** を設定する前に **event_wait** を設定した場合のエラーを示しています。

次に、**event_completion** Tcl コマンドと**event_wait** コマンドの両方を使用した出力例を示します。

```
::cisco::eem::event_register_syslog tag e1 occurs 1 pattern CLEAR maxrun 0
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
set i 1
while {1 == 1} {  # Start high performance policy loop
   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 msg "event $i serviced" priority info
   if {$i == 5} {
       action_syslog msg "Exiting after servicing 5 events" priority info
       exit 0
   }
   incr i
   event_completion status 0
   array set _event_state_arr [event_wait refresh_vars 0]
   if {$_event_state_arr(event_state) != 0} {
       action_syslog msg "Exiting: failed event_state " \
                         " $event_state_arr(event_state)" priority info
       exit 0
   }
}
```

次に、実行コンフィギュレーションの例を示します。

```
Device#
01:00:44: %SYS-5-CONFIG_I: Configured from console by consoleclear counters
Clear "show interface" counters on all interfaces [confirm]
Device#
01:00:49: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:00:49: %HA_EM-6-LOG: high_perf_example.tcl: event 1 serviced
Device#
Device#clear counters
Clear "show interface" counters on all interfaces [confirm]
Device#
Device#
01:00:53: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:00:53: %HA_EM-6-LOG: high_perf_example.tcl: event 2 serviced
Device#clear counters
Clear "show interface" counters on all interfaces [confirm]
Device#
Device#
01:00:56: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:00:56: %HA_EM-6-LOG: high_perf_example.tcl: event 3 serviced
Device#
Device#
Device#clear counters
Clear "show interface" counters on all interfaces [confirm]
Device#
01:00:59: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
Device#
```

```
01:00:59: %HA_EM-6-LOG: high_perf_example.tcl: event 4 serviced
01:00:59: %HA_EM-6-LOG: high_perf_example.tcl: Exiting after servicing 5 events
Device#
Device#
Device#copy tftp disk1:
Address or name of remote host [dirt]?
Source filename [user/eem_scripts/high_perf_example.tcl]?
Destination filename [high_perf_example.tcl]?
%Warning:There is a file already existing with this name
Do you want to over write? [confirm]
Accessing tftp://dirt/user/eem_scripts/high_perf_example.tcl...
Loading user/eem_scripts/high_perf_example.tcl from 192.0.2.19 (via FastEthernet0/0): !
[OK - 909 bytes]
909 bytes copied in 0.360 secs (2525 bytes/sec)
Device#
Device#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Device(config)#no event manager policy high_perf_example.tcl
Device(config)#event manager po high_perf_example.tcl
Device(config)#end
Device#
Device#
Device#
Device#
01:02:19: %SYS-5-CONFIG_I: Configured from console by consoleclear counters
Clear "show interface" counters on all interfaces [confirm]
Device#
01:02:23: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
Device#
Device#
01:02:23: %HA_EM-6-LOG: high_perf_example.tcl: event 1 serviced
Device#
Device#clear counters
Clear "show interface" counters on all interfaces [confirm]
Device#
Device#
01:02:26: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:02:26: %HA_EM-6-LOG: high_perf_example.tcl: event 2 serviced
Device#
Device#clear counters
Clear "show interface" counters on all interfaces [confirm]
Device#
Device#
01:02:29: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:02:29: %HA_EM-6-LOG: high_perf_example.tcl: event 3 serviced
Device#
Device#clear counters
Clear "show interface" counters on all interfaces [confirm]
Device#
Device#
01:02:33: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
Device#
01:02:33: %HA_EM-6-LOG: high_perf_example.tcl: event 4 serviced
Device#
Device#clear counters
Clear "show interface" counters on all interfaces [confirm]
Device#
Device#
Device#
01:02:36: %CLEAR-5-COUNTERS: Clear counter on all interfaces by console
01:02:36: %HA_EM-6-LOG: high_perf_example.tcl: event 5 serviced
01:02:36: %HA_EM-6-LOG: high_perf_example.tcl: Exiting after servicing 5 events
Device#
```

また、イベントがサービスされ、次のイベントの到着を待っている間に、**show event manager policy active** コマンドによって、次の出力が表示されます。

```
Device#show event manager policy active
Key: p - Priority        :L - Low, H - High, N - Normal, Z - Last
     s - Scheduling node :A - Active, S - Standby
default class - 1 script event
 no.  job id      p s status  time of event             event type          name
 1    11          N A wait    Mon Oct20 14:15:24 2008   syslog
high_perf_example.tcl
```

前述の例では、ステータスは待ち状態です。これは、ポリシーが次のイベントの到着を待っていることを示します。

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