EEM CLI ライブラリ XML-PI サポート

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

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

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

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

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

xml_pi_exec

fd 引数を使用してハンドラが指定され、spec_file 引数によって spec-file が指定されてコマンドが実行されるチャネルに対して、cmd 引数を使用して指定された XML-PI コマンドを書き込みます。コマンドの未加工XML出力データが、チャネルから読み取られ、XML出力が返されます。

構文

```
xml_pi_show fd cmd [spec_file]
```

引数

fd	（必須）cli_open から取得された CLI ライブラリ ファイル記述子。
cmd	（必須）IOS 表示コマンド。
spec_file	（任意）IOS CLI 表示コマンド spec_file。

結果文字列

IOS 表示コマンドの結果（XML フォーマット）。

_cerrno を設定

発生する可能性があるエラーは、次のとおりです。

1.チャンネル読み取りエラー

xml_pi_parse

この機能に xml_data として渡される XML 表示コマンドの未加工出力を処理し、xml_tags_list によって指定されたこれらのフィールドを取得します。次の処理が発生します。

ステップ 1：XML タグ リストは、Tcl リストとして有効にされます。低い順序の名前が、該当するコマンドであいまいな場合、XML タグは、低い順序の XML タグ名または完全修飾 XML タグ名として指定できます。

タグ例：<Interface> <ShowIpInterfaceBrief><IPInterfaces><entry><Interface>

ステップ 2：xml_data は、有効な XML として有効にされ、XML パース ツリーに解釈されます。

ステップ 3：XML パース ツリーをウォークし、各タグが、XML タグ リストのエントリと比較されます。照会が発生する場合、タグ名が、現在の Tcl スコープ内で定義されている Tcl 手順と一致しているかどうかが判断されます。一致する場合、Tcl 手順は、現在の結果で呼び出さ

れます。一致しない場合、タグ名と、タグ名に関連付けられているデータは、現在の結果の末尾に追加されます。

構文

```
xml_pi_parse fd xml_show_cmd_output xml_tags_list
```

引数

fd	（必須）cli_open から取得された CLI ライブラリ ファイル記述子。
xml_show_cmd_output	（必須）xml 形式での、xml_pi_show コマンド拡張の出力。
xml_tags_list	（必須）関連するタグのリスト。

結果文字列

XML タグ名によって索引化される Tcl 配列のデータ。

（注） 現在の結果は、Tcl 手順の呼び出し後にリセットされます。

_cerrno を設定

発生する可能性があるエラーは、次のとおりです。

1. XML タグリストの分割中にエラーが発生する 2. XML タグリストが null に指定されている 3. XML タグツリーが 20 レベルを超えている 4. 呼び出された Tcl プロシージャがエラーを返した 5. メモリ割り当ての失敗 6. XML 解析エラー 7. XML ドメインの作成に失敗

xml_pi_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、ファイル記述子によって指定されているハンドラがある CLI チャネルから（指定された表示コマンドから）XML-PI コマンド出力を読み取ります。XML 形式で、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

```
xml_pi_read fd
```

引数

fd	（必須）cli_open から取得された CLI ライブラリ ファイル記述子。

結果文字列

XML 形式で読み取られるすべての内容。

_cerrno を設定

発生する可能性があるエラーは、次のとおりです。

1. ルータ名を取得できない 2. コマンドエラー

xml_pi_write

fd 引数を使用してハンドラが指定され、spec_file 引数によって仕様ファイルが指定されてコマンドが実行されるチャネルに対して、cmd 引数を使用して指定された XML-PI コマンドを書き込みます。

構文

```
xml_pi_write fd cmd spec_file
```

引数

fd	（必須）cli_open から取得された CLI ライブラリ ファイル記述子。
cmd	（必須）IOS 表示コマンド。
spec_file	（任意）IOS CLI 表示コマンド spec_file。

結果文字列

なし

_cerrno を設定

なし

XML-PI 機能のサンプル使用

次の EEM ポリシー（sample.tcl）で、新しい EEM XML-PI 機能の 5 つの異なる実装の 1 つの例を示します。odm spec-file（例 2）が、このポリシーに続きます。

```
::cisco::eem::event_register_none maxrun 60
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
# open the cli_lib.tcl channel
if [catch {cli_open} result] {
error $result $errorInfo
} else {
array set cli1 $result
}
```

```
# enter "enable" privilege mode
if [catch {cli_exec $cli1(fd) "en"} result] {
error $result $errorInfo
}
# Example 1:
#
# Detect if XML-PI is present in this image
# Invoke xml_pi_exec with the default spec file for the "show inventory"
# command. After the command executes $result contains the raw XML data if
# the command is successful.
if [catch {xml_pi_exec $cli1(fd) "show inventory" ""} result] {
puts "Example 1: XML-PI support is not present in this image - exiting"
exit
} else {
puts "Example 1: XML-PI support is present in this image"
}
# Example 2:
#
# In the next example we demonstrate how to extract two data elements
# from the "show version" command using the specified XML-PI spec file.
# The raw output from this command is as follows:
#
# Device#show version | format disk2:speceemtest.odm
# <?xml version="1.0" encoding="UTF-8"?>
# <ShowVersion>
# <Version>12.4(20071029:194217)</Version>
# <Compiled>Thu 08-Nov-07 11:28</Compiled>
# <ROM>System Bootstrap, Version 12.2(20030826:190624) [BLD-npeg1_rommon_r11 102],
DEVELOPMENT</ROM>
# <uptime>17 minutes</uptime>
# <processor>NPE-G1</processor>
# <bytesofmemory>983040K/65536K</bytesofmemory>
# <CPU>700MHz</CPU>
# <L2Cache>0.2</L2Cache>
# <GigabitEthernetinterfaces>3</GigabitEthernetinterfaces>
# <bytesofNVRAM>509K</bytesofNVRAM>
# <bytesofATAPCMCIAcard>125952K</bytesofATAPCMCIAcard>
# <Sectorsize>512 bytes</Sectorsize>
# <bytesofFlashinternalSIMM>16384K</bytesofFlashinternalSIMM>
# <Configurationregister>0x2100</Configurationregister>
# </ShowVersion>
#
# Invoke xml_pi_exec with the spec file "disk2:speceemtest.odm" for the
# "show version" command. After the command executes $result contains
# the raw XML data.
if [catch {xml_pi_exec $cli1(fd) "show version" "disk2:speceemtest.odm"} result] {
error $result $errorInfo
} else {
# Pass the raw XML data to the xml_pi_parse routine to extract fields
# of interest:
# we ask that only the <processor> and <CPU> fields be returned.
array set xml_result [xml_pi_parse $cli1(fd) $result "<processor> <CPU>"]
puts "Example 2: Processor is $xml_result(<processor>) CPU is $xml_result(<CPU>)"
}
# Example 3:
#
# In the next example we demonstrate how to extract two data elements
# from the multi-record "show inventory" command using the default built-in
# XML-PI spec file. Sample raw output from this command is as follows:
#
# Device#show inventory | format
# <?xml version="1.0" encoding="UTF-8"?>
# <ShowInventory>
# <SpecVersion>built-in</SpecVersion>
```

```
# <InventoryEntry>
# <ChassisName>"Chassis"</ChassisName>
# <Description>"Cisco 7206VXR, 6-slot chassis"</Description>
# <PID>CISCO7206VXR</PID>
# <VID>
# </VID>
# <SN>31413378 </SN>
# </InventoryEntry>
# <InventoryEntry>
# <ChassisName>"NPE-G1 0"</ChassisName>
# <Description>"Cisco 7200 Series Network Processing Engine
NPE-G1"</Description>
# <PID>NPE-G1</PID>
# <VID>
# </VID>
# <SN>31493825 </SN>
# </InventoryEntry>
# <InventoryEntry>
# <ChassisName>"disk2"</ChassisName>
# <Description>"128MB Compact Flash Disk for NPE-G1"</Description>
# <PID>MEM-NPE-G1-FLD128</PID>
# <VID>
# </VID>
# <SN>NAME: "module 1"</SN>
# </InventoryEntry>
# <InventoryEntry>
# <ChassisName>"module 1"</ChassisName>
# <Description>"Dual Port FastEthernet (RJ45)"</Description>
# <PID>PA-2FE-TX</PID>
# <VID>
# </VID>
# <SN>JAE0827NGKX</SN>
# </InventoryEntry>
# <InventoryEntry>
# <ChassisName>"Power Supply 2"</ChassisName>
# <Description>"Cisco 7200 AC Power Supply"</Description>
# <PID>PWR-7200-AC</PID>
# <VID>
# </VID>
# </InventoryEntry>
# </ShowInventory>
#
# Define a procedure to be called every time the <InventoryEntry> tag
# is processed. Since this tag precedes each new output record, the data
# that is passed into this procedure contains the fields that have been
# requested via xml_pi_parse since the previous time this procedure was
# called.
proc <InventoryEntry> {xml_line} {
global num
# The first time that this function is called there is no data and
# xml_line will be null.
if [string length $xml_line] {
array set xml_result $xml_line
incr num
set output [format "Example 3: Item %2d %-18s %s" \
$num $xml_result(<PID>) $xml_result(<Description>)]
puts $output
}
}
set num 0
# Invoke xml_pi_exec with the default built-in spec file for the
# "show inventory" command. After the command executes $result contains
# the raw XML data.
if [catch {xml_pi_exec $cli1(fd) "show inventory"} result] {
```

```
error $result $errorInfo
} else {
# Pass the raw XML data to the xml_pi_parse routine to extract fields
# of interest:
# we ask that only the <PID> and <Description> fields be returned.
# If an XML tag name is requested and a Tcl proc exists with that name,
# the Tcl proc will be called every time that tag is encountered in the
# output data. Specify the <InventoryEntry> tag and define the proc
# before executing the xml_pi_parse statement.
array set xml_result [xml_pi_parse $cli1(fd) $result \
"<InventoryEntry> <PID> <Description>"]
# Display the data from the last record.
incr num
set output [format "Example 3: Item %2d %-18s %s" \
$num $xml_result(<PID>) $xml_result(<Description>)]
puts $output
}
# Example 4:
#
# In the next example we demonstrate how to extract two data elements
# from the multi-record "show ip interface brief" command using the default
# built-in XML-PI spec file. Sample raw output from this command is as
# follows:
#
# Device#show ip interface brief | format
# <?xml version="1.0" encoding="UTF-8"?>
# <ShowIpInterfaceBrief>
# <SpecVersion>built-in</SpecVersion>
# <IPInterfaces>
# <entry>
# <Interface>GigabitEthernet0/1</Interface>
# <IP-Address>172.19.209.34</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>up</Status>
# <Protocol>up</Protocol>
# </entry>
# <entry>
# <Interface>GigabitEthernet0/2</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# <entry>
# <Interface>GigabitEthernet0/3</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# <entry>
# <Interface>FastEthernet1/0</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# <entry>
# <Interface>FastEthernet1/1</Interface>
# <IP-Address>unassigned</IP-Address>
```

```
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# </IPInterfaces>
# </ShowIpInterfaceBrief>
#
# Define a procedure to be called every time the fully qualified name
# <ShowIpInterfaceBrief><IPInterfaces><entry> tag is processed. Since
# this tag precedes each new output record, the data that is passed into
# this procedure contains the fields that have been requested via
# xml_pi_parse since the previous time this procedure was called.
proc <ShowIpInterfaceBrief><IPInterfaces><entry> {xml_line} {
global num
# The first time that this function is called there is no data and
# xml_line will be null.
if [string length $xml_line] {
array set xml_result $xml_line
incr num
set output [format "Example 4: Interface %2d %-30s %s" \
$num $xml_result(<Interface>) $xml_result(<Status>)]
puts $output
} else {
puts "Example 4: Display All Interfaces"
}
}
set num 0
# Invoke xml_pi_exec with the default built-in spec file for the
# "show ip interface brief" command. After the command executes $result
# contains the raw XML data.
if [catch {xml_pi_exec $cli1(fd) "show ip interface brief"} result] {
error $result $errorInfo
} else {
# Pass the raw XML data to the xml_pi_parse routine to extract fields
# of interest:
# we ask that only the <Interface> and <Status> fields be returned.
# If an XML tag name is requested and a Tcl proc exists with that name,
# the Tcl proc will be called every time that tag is encountered in the
# output data. Specify the <entry> tag and define the proc
# before executing the xml_pi_parse statement.
array set xml_result [xml_pi_parse $cli1(fd) $result \
"<ShowIpInterfaceBrief><IPInterfaces><entry> <Interface> <Status>"]
# Display the data from the last record.
incr num
set output [format "Example 4: Interface %2d %-30s %s" \
$num $xml_result(<Interface>) $xml_result(<Status>)]
puts $output
}
# Example 5:
#
# In the next example we demonstrate how to extract two data elements
# from the multi-record "show ip interface brief" command using the default
# built-in XML-PI spec file. Sample raw output from this command is as
# follows:
#
# Device#show ip interface brief | format
# <?xml version="1.0" encoding="UTF-8"?>
# <ShowIpInterfaceBrief>
# <SpecVersion>built-in</SpecVersion>
# <IPInterfaces>
# <entry>
# <Interface>GigabitEthernet0/1</Interface>
# <IP-Address>172.19.209.34</IP-Address>
```

```
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>up</Status>
# <Protocol>up</Protocol>
# </entry>
# <entry>
# <Interface>GigabitEthernet0/2</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# <entry>
# <Interface>GigabitEthernet0/3</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# <entry>
# <Interface>FastEthernet1/0</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# <entry>
# <Interface>FastEthernet1/1</Interface>
# <IP-Address>unassigned</IP-Address>
# <OK>YES</OK>
# <Method>NVRAM</Method>
# <Status>administratively down</Status>
# <Protocol>down</Protocol>
# </entry>
# </IPInterfaces>
# </ShowIpInterfaceBrief>
#
# Note: This example is the same as Example 4 with the exception that
# the new record procedure is called by the un-qualified tag name. The
# ability to specify the un-qualified tag names is simplier but only works
# if the un-qualified name is used once per Tcl program. In this example
# the unqualified new record tag name is "<entry>" which is a very
# common name in the Cisco spec file.
# Define a procedure to be called every time the <entry> tag
# is processed. Since this tag precedes each new output record, the data
# that is passed into this procedure contains the fields that have been
# requested via xml_pi_parse since the previous time this procedure was
# called.
proc <entry> {xml_line} {
global num
# The first time that this function is called there is no data and
# xml_line will be null.
if [string length $xml_line] {
array set xml_result $xml_line
incr num
if ([string equal $xml_result(<Status>) "up"]) {
set output [format "Example 5: Interface %2d %-30s %s" \
$num $xml_result(<Interface>) $xml_result(<Status>)]
puts $output
}
} else {
```

```
puts "Example 5: Display All Interfaces That Are Up"
}
}
set num 0
# Invoke xml_pi_exec with the default built-in spec file for the
# "show ip interface brief" command. After the command executes $result
# contains the raw XML data.
if [catch {xml_pi_exec $cli1(fd) "show ip interface brief"} result] {
error $result $errorInfo
} else {
# Pass the raw XML data to the xml_pi_parse routine to extract fields
# of interest:
# we ask that only the <Interface> and <Status> fields be returned.
# If an XML tag name is requested and a Tcl proc exists with that name,
# the Tcl proc will be called every time that tag is encountered in the
# output data. Specify the <entry> tag and define the proc
# before executing the xml_pi_parse statement.
array set xml_result [xml_pi_parse $cli1(fd) $result \
"<entry> <Interface> <Status>"]
# Display the data from the last record.
incr num
if ([string equal $xml_result(<Status>) "up"]) {
set output [format "Example 5: Interface %2d %-30s %s" \
$num $xml_result(<Interface>) $xml_result(<Status>)]
puts $output
}
}
```

XML-PI 仕様 eemtest.odm ODM ファイルの例

```
###
show version
<?xml version='1.0' encoding='utf-8'?>
<ODMSpec>
<Command>
<Name>show version</Name>
</Command>
<OS>ios</OS>
<DataModel>
<Container name="ShowVersion">
<Property name="Version" distance = "1.0" length = "1" type = "IpAddress"/>
<Property name="Technical Support" distance = "1.0" length = "1" type = "IpAddress"/>
<Property name="Compiled" distance = "1.0" length = "3" type = "String"/>
<Property name="ROM" distance = "1.0" length = "7" type = "IpAddress"/>
<Property name="uptime" distance = "2" length = "8" type = "String"/>
<Property name="image" distance = "4" length = "1" type = "IpAddress"/>
<Property name="processor" distance = "-1" length = "1" type = "String"/>
<Property name="bytes of memory" distance = "-1" length = "1" type = "Port"/>
<Property name="CPU" distance = "2" length = "1" end-delimiter = "," type = "String"/>
<Property name="L2 Cache" distance = "-2" length = "1" end-delimiter = "," type =
"String"/>
<Property name="Gigabit Ethernet interfaces" distance = "-1" length = "1" type =
"Integer"/>
<Property name="bytes of NVRAM" distance = "-1" length = "1" type = "String"/>
<Property name="bytes of ATA PCMCIA card" distance = "-1" length = "1" type = "String"/>
<Property name="Sector size" distance = "1.0" length = "2" end-delimiter = ")" type =
"String"/>
<Property name="bytes of Flash internal SIMM" distance = "-1" length = "1" type =
"String"/>
<Property name="Configuration register" distance = "2" length = "1" type = "String"/>
</Container>
</DataModel>
</ODMSpec>
```

sample.tcl の実行例

```
Device#config t
Enter configuration commands, one per line. End with CNTL/Z.
Device(config)#event manager policy sample.tcl
Device(config)#end
Device#
Oct 10 20:21:26: %SYS-5-CONFIG_I: Configured from console by console
Device#event manager run sample.tcl
Example 1: XML-PI support is present in this image
Example 2: Processor is NPE-G1 CPU is 700MHz
Example 3: Item 1 CISCO7206VXR "Cisco 7206VXR, 6-slot chassis"
Example 3: Item 2 NPE-G1 "Cisco 7200 Series Network Processing Engine NPE-G1"
Example 3: Item 3 MEM-NPE-G1-FLD128 "128MB Compact Flash Disk for NPE-G1"
Example 3: Item 4 PA-2FE-TX "Dual Port FastEthernet (RJ45)"
Example 3: Item 5 PWR-7200-AC "Cisco 7200 AC Power Supply"
Example 4: Display All Interfaces
Example 4: Interface 1 GigabitEthernet0/1 up
Example 4: Interface 2 GigabitEthernet0/2 administratively down
Example 4: Interface 3 GigabitEthernet0/3 administratively down
Example 4: Interface 4 FastEthernet1/0 administratively down
Example 4: Interface 5 FastEthernet1/1 administratively down
Example 4: Interface 6 SSLVPN-VIF0 up
Example 5: Display All Interfaces That Are Up
Example 5: Interface 1 GigabitEthernet0/1 up
Example 5: Interface 6 SSLVPN-VIF0 up
```

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。