UVEXPLORER CUSTOMIZATION

CUSTOMIZING UVEXPLORER TO WORK IN YOUR ENVIRONMENT

JUNE 21, 2018

UVNETWORKS

Contents

Customizing UVexplorer

UVexplorer specializes in discovering network devices, and provides built-in support for many of the device types most commonly found in IT environments. However, the variety of different device types used in the world is virtually limitless, and therefore it is impossible for a product such as UVexplorer to have detailed knowledge of every possible device type. For this reason, you can add intelligence to UVexplorer regarding new device types that it does not know much about. This document describes how this is done.

Adding Support for New Device Types

Suppose you have a device in your IT environment that UVexplorer does not have detailed knowledge about. There are several things you can do to configure UVexplorer so it can work more effectively with your device.

1. **Custom Device Descriptions, Vendors, and Categories:** When UVexplorer discovers your network, it tries to identify each device on your network, and for each device determine what the device is, the vendor who made it, and what categories the device belongs to (router, switch, firewall, etc.). If UVexplorer is unable to determine this information for one of your devices, you can configure it with the necessary information so it will be able to identify your device.
2. **Custom Network Map Images:** When UVexplorer draws a network map, each device on the map is drawn with an image that depicts the device's type (router, switch, firewall, etc.). UVexplorer allows you to customize the images it uses to draw devices. This is useful if you want to change the default images used by UVexplorer, or provide device images for new device types that you define.
3. **Custom SSH/Telnet Settings:** UVexplorer is capable of using the SSH and Telnet protocols to discover information about devices. Using these protocols, UVexplorer can log in to devices, run shell commands, and capture command outputs, thus collecting information about devices. To correctly log in to a device, UVexplorer requires several configuration settings that allow it to properly interact with the device (user name prompt, password prompt, command-line prompt, etc.). These settings vary from device to device. UVexplorer's built-in settings will work for many devices, but for some devices you may need to provide custom settings.
4. **Custom Device Configuration Command Scripts:** As mentioned in the previous item, UVexplorer can log in to devices using the SSH or Telnet protocols, run shell commands, and capture the output of those commands. One of the most important uses of this capability is to capture device configurations, and store them in the UVexplorer database. The shell commands required to capture device configurations vary from device to device. UVexplorer has built-in scripts that will successfully capture configurations of many devices, but for some devices you may need to provide custom command scripts that UVexplorer can use to capture device configurations.

Custom Device Descriptions, Vendors, and Categories

If UVexplorer is unable to determine the description, vendor, or device categories for one of your devices, you can specify some or all of this information by creating an XML file with the following name and location:

\ProgramData\UVnetworks\UVexplorer\2.0**user-key-classifier-map.xml**

The root element of this file should be a `<key-classifier-map>` element. Within the root `<key-classifier-map>` element you can place any number of `<e>` elements, where "e" stands for entry.

```
<key-classifier-map>
  <e> … </e>
  <e> … </e>
  <e> … </e>
  …
</key-classifier-map>
```

The format of each entry element is as follows:

```
  <e>
    <key> … </key>
    <descr>Device Description Text</descr>
    <cats>Comma-separated list of device category strings</cats>
    <ven>Vendor name</ven>
  </e>
```

Every device in the network that matches the entry's `<key>` will be assigned the specified description, device categories, and vendor name. While the `<key>` sub-element is required, each of the `<descr>`, `<cats>`, and `<ven>` sub-elements are optional. You can specify only the pieces of information that matter to you.

The <key> sub-element (required)

The `<key>` sub-element specifies which devices an entry applies to. The `<key>` may specify the device's MAC address, IP address, or SNMP OID.

For MAC address keys, the key may specify a complete MAC address, or only a prefix. If the key specifies a complete MAC address, only the device with that MAC address will match the entry. If the key specifies a MAC address prefix, all devices with MAC addresses that have the specified prefix will match the entry.

```
<!-- Key defined by MAC address -->
<key>
  <e>
    <typ>mac</typ>
    <val>2c3033</val>
  </e>
</key>
```

For IP address keys, the key must specify a complete IP address. Devices with the specified IP address will match the entry.

```
<!-- Key defined by IP address -->
<key>
  <e>
    <typ>ip</typ>
    <val>192.168.1.1</val>
  </e>
</key>
```

For SNMP OID keys, the key may specify a complete OID, or only a prefix. If the key specifies a complete OID, only devices with that OID will match the entry. If the key specifies an OID prefix, all devices with OIDs that have the specified prefix will match the entry.

```
<!-- Key defined by SNMP OID -->
<key>
  <e>
    <typ>oid</typ>
    <val>1.3.6.1.4.1.9</val>
  </e>
</key>
```

You can also create keys that contain multiple entries. For example, a key might specify multiple MAC addresses, or multiple IP addresses, or multiple OIDs (or any combination of these). An entry with a multi-value key will match all devices that match any of the values specified in the key. For example, the following key will match any device whose MAC address, IP address, or OID match the specified values:

```
<!-- Key with multiple entries -->
<key>
  <e>
    <typ>mac</typ>
    <val>2c3033</val>
  </e>
  <e>
    <typ>ip</typ>
    <val>192.168.1.1</val>
  </e>
  <e>
    <typ>oid</typ>
    <val>1.3.6.1.4.1.9</val>
  </e>
</key>
```

The <descr> sub-element (optional)

The `<descr>` sub-element contains any text you want that describes the device. For example,

```
<descr>Catalyst 2916M-XL switch</descr>
```

The <ven> sub-element (optional)

The <ven> sub-element contains the name of the device's vendor. For example,

```
<ven>Cisco</ven>
```

UVexplorer has built-in support for the following vendors. You are free to add additional vendors as your needs require.

Vendor Name
Cisco
Juniper
Hewlett Packard
3Com
Apple
IBM
Microsoft
Dell
Foundry
Nortel
Aruba
Extreme
Avaya
Enterasys
VMware
Motorola

The <cats> sub-element (optional)

The <cats> sub-element contains a comma-separated list of device category names. For example,

```
<cats>router, switch, net-device</cats>
```

UVexplorer has a long list of built-in device category names that you can use (e.g., router, switch, firewall, etc.). Or, you can specify your own custom device category names (e.g., my-custom-category). The built-in categories are listed below:

Device Category Name	Meaning
snmp	The device supports the SNMP protocol
net-device	The device is a network device
core-device	The device is part of the core network infrastructure
server	The device is a server
workstation	The device is a workstation
printer	The device is a printer
switch	The device is a switch
router	The device is a router
firewall	The device is a firewall
wireless-controller	The device is a wireless access point controller

wireless-ap	The device is a wireless access point
wireless-client	The device is connected to a wireless access point
wireless-rogues	The device is a rogue (i.e., unauthorized) wireless device
ip-phone-manager	The device is an IP phone manager
ip-phone	The device is an IP phone
windows	The device is running the Microsoft Windows operating system
windows-server	The device is running the Microsoft Windows Server operating system
apple	The device is an Apple device
linux	The device is running the Linux operating system
unix	The device is running the Unix operating system
laptop	The device is a laptop
virtual	The device is virtual
virtual-host	The device is a virtual host (i.e., hosts virtual machines)
virtual-machine	The device is a virtual machine
virtual-server	The device is a virtual server (i.e., controls virtual hosts)
vmware	The device is running VMware
hyper-v	The device is running Microsoft Hyper-V
vsphere	The device is running VMware Vsphere
vcenter	The device is running VMware Vcenter
virtual-switch	The device is a virtual switch
ms-active-directory	The device is running Microsoft Active Directory
ms-exchange	The device is running Microsoft Exchange
ms-iis	The device is running Microsoft IIS
ms-sqlserver	The device is running Microsoft SQL Server
dhcp	The device is acting as a DHCP server
ldap	The device is acting as an LDAP server
smtp	The device is acting as an SMTP server
imap	The device is acting as an IMAP server
pop	The device is acting as a POP server

Example File

Here is an example `user-key-classifier-map.xml` file:

```
<key-classifier-map>

  <!-- Entry that identifies Netgear devices by MAC address prefix -->
  <e>
    <key>
      <e>
        <typ>mac</typ>
        <val>2c3033</val>
      </e>
    </key>
    <ven>NETGEAR</ven>
  </e>
```

```
  <!-- Entry that identifies a device by IP address -->
  <e>
    <key>
      <e>
        <typ>ip</typ>
        <val>192.168.1.1</val>
      </e>
    </key>
    <descr>My Custom Device</descr>
    <cats>net-device, core-device, my-custom-device</cats>
    <ven>ACME Enterprises</ven>
  </e>

  <!-- Entry that identifies Cisco devices by SNMP OID prefix -->
  <e>
    <key>
      <e>
        <typ>oid</typ>
        <val>1.3.6.1.4.1.9</val>
      </e>
    </key>
    <ven>Cisco</ven>
  </e>

</key-classifier-map>
```

Custom Network Map Images

When UVexplorer draws a device on a network map, it displays the device using the image that best represents the device's primary device category. While a device may belong to many device categories, typically one of these categories most closely represents the essential nature of the device. For example, a network switch device might belong to the following device categories: switch, net-device, core-device, snmp. Of these categories, "switch" most directly captures the essential nature of the device. Thus, UVexplorer would render this device on a network map using the image of a switch.

Additionally, for each device on a network map, if UVexplorer knows the vendor of the device, it will overlay an image representing the device vendor's logo on the corner of the device's image. Therefore, two images are needed to render each device on a map: 1) the image that represents the device itself, and 2) the image that represents the vendor's logo.

The previous section explained how to configure UVexplorer to correctly identify and categorize device types that it does not natively support. You can even define new device categories and device vendors of which UVexplorer was previously unaware. When you do this, it is often desirable to also add new images for drawing new device categories and vendors on network maps. If you need to expand UVexplorer's built-in collection of device category and vendor images, or override UVexplorer's default built-in images, do the following:

1. Create an XML file with the following name and location:
   \ProgramData\UVnetworks\UVexplorer\2.0**user-topo-image-map.xml**

2. Create a folder with the following name and location in which to put your custom images: \ProgramData\UVnetworks\UVexplorer\2.0**user-topo-images**

The root element of the `user-topo-image-map.xml` file should be a `<tpimgmap>` element. `<tpimgmap>` stands for "topology image map". Within the root `<tpimgmap>` element you can place any number of `<e>` elements, where "e" stands for entry.

```
<tpimgmap>
  <e> … </e>
  <e> … </e>
  <e> … </e>
  …
</tpimgmap>
```

There are two kinds of entries:

1. Entries that specify the image to use for a particular device category
2. Entries that specify the image to use for a particular vendor

The following example shows a device category entry. This entry matches all devices with device category "router".

```
  <e>
       <tpimgk>
            <e>
              <name>dvcat</name>
              <val>router</val>
            </e>
       </tpimgk>
       <flnm>router.png</flnm>
  </e>
```

The next example shows a vendor entry. This entry matches all devices with vendor "Cisco".

```
  <e>
       <tpimgk>
            <e>
              <name>vnd</name>
              <val>Cisco</val>
            </e>
       </tpimgk>
       <flnm>vendor-images\cisco.png</flnm>
  </e>
```

The `<tpimgk>` sub-element contains the entry's "key", and the `<flnm>` sub-element contains the name of the entry's image file.

The <tpimgk> sub-element (required)

`<tpimgk>` stands for "topology image key". This sub-element contains the entry's "key", and specifies which device category or vendor the entry applies to. For device category entries, all devices with the specified device category will be drawn using the entry's image. For vendor entries, all devices with the specified vendor will be annotated with the entry's image.

This is an example of a device category key. This key matches all devices with the "router" device category.

```
<tpimgk>
     <e>
       <name>dvcat</name>
       <val>router</val>
     </e>
 </tpimgk>
```

This is an example of a vendor key. This key matches all devices with vendor "Cisco".

```
<tpimgk>
     <e>
       <name>vnd</name>
       <val>Cisco</val>
     </e>
 </tpimgk>
```

The <flnm> sub-element (required)

`<flnm>` stands for "file name". This sub-element specifies the entry's image file.

All image files should be stored in the following directory:

\ProgramData\UVnetworks\UVexplorer\2.0**user-topo-images**

The value of the `<flnm>` sub-element is the path of the entry's image file relative to the `user-topo-images` folder. The following examples demonstrate what this looks like:

```
<flnm>router.png</flnm>

<flnm>vendor-images\cisco.png</flnm>
```

Image Files

All image files should be in the PNG format. Device category images should be 48 x 48 pixels in size. Vendor images should be 24 x 24 pixels in size.

Custom SSH/Telnet Settings

In order to interact with a device through the SSH or Telnet protocols, UVexplorer must know several things about how the command-line interface of the device works. For example, what prompt does the device display when prompting for the user's login name? What is the device's password prompt? What is the device's command prompt? Etc. UVexplorer calls these the "command-line settings" for

the device. Additionally, for a given device, the SSH and Telnet command-line interfaces might work differently, and require different settings. UVexplorer has default command-line settings for both SSH and Telnet which will work for many devices. If more specific settings for a device are not available, UVexplorer will try to use these default settings to interact with the device. However, these defaults do not always work, and you may need to provide specific settings for some of the devices on your network. (Note, these settings are only needed for devices that UVexplorer interacts with using SSH or Telnet.)

Command-Line Settings

When a device is accessed using a command-line protocol such as SSH or Telnet, the first step is to establish a connection with the device. Once a connection has been established, the second step is to log in to the device by providing a user name and password. After log in is complete, the third step is to run a command by typing it on the command-line, pressing enter, and viewing the output of the command.

To perform this basic interaction, there are several settings that UVexplorer needs to successfully interact with the device:

1. The “user name prompt” displayed by the device. UVexplorer needs to know what this is so that it knows when to send the user name to the device.
2. The “password prompt” displayed by the device. UVexplorer needs to know what this is so that it knows when to send the password to the device.
3. The “command prompt” displayed by the device. UVexplorer needs to know what this is so that it knows when to send the command to the device.

For some devices, when a command that requires additional privileges is executed, the device will prompt the user for an “enable password”, which is the password required to elevate to a higher privilege level. In this case, UVexplorer needs to know the “enable password prompt” displayed by the device so it knows when to send the enable password to the device.

For many devices, when the output of a command is too long to fit on a single screen, the device displays the command output one page at a time, and requires the user to press a key in order to see the next page of output. In this case, UVexplorer needs to know the “more prompt” displayed by the device when it is waiting for the “next page” input from the user, and also the input that should be sent to the device to advance to the next page (the “more response”). [Note: On many devices there is a command you can execute to disable paged output. If such a command exists, it is usually best to disable paged output before running other commands that might produce long outputs. This makes it easier for UVexplorer to correctly capture the output of such commands.]

Some devices immediately display a menu when you connect to them, and you must select a menu option before logging in with user name and password. When this is the case, UVexplorer needs to know the “login menu prompt” displayed by the device, and also the input it should send to the device in order to select the correct menu option (the “login menu response”).

Terminal Types

When interacting with devices through SSH or Telnet, devices may return control character sequences that are intended to control the user’s interactive terminal (“clear the screen”, “scroll the screen”, etc.).

While such control sequences are useful for people using a terminal to interactively access a device, they are not useful for UVexplorer, because it is not using an interactive terminal at all. For this reason, UVexplorer provides a command-line setting named "terminal type" that allows you to control the type of terminal it should emulate when accessing a device through SSH or Telnet. Typically, it is best to use a terminal type that turns off control character sequences from the device, such as "tty" or "std". By default, UVexplorer uses a terminal type of "tty", but you can change this through the "terminal type" command-line setting.

Regular Expressions

The values of the various command-line settings are defined using literal character strings, but some settings' values can also be defined using regular expressions. Regular expressions are more powerful than literal strings, and make it possible to create settings that work even when the exact output expected from a device is unknown, but can be generally characterized with a regular expression.

For settings that support regular expressions, you can specify their values using either a literal string or a regular expression. To distinguish a regular expression from a literal string, regular expressions must be prefixed with the string "regex:". For example, "regex:.*> ?$" is a regular expression that will match a sequence of zero or more of any character followed by a greater-than sign followed by an optional space. UVexplorer uses the Perl Compatible Regular Expressions (PCRE) library to process regular expressions, so you can reference the documentation for the PCRE library for details on regular expression syntax (www.pcre.org).

Specifying Custom SSH/Telnet Settings

If you need to specify custom command-line settings for a device, you can do so by creating an XML file with the following name and location:

\ProgramData\UVnetworks\UVexplorer\2.0**user-clsettings-map.xml**

The root element of this file should be a `<clsettings-map>` element. Within the root `<clsettings-map>` element you can place any number of `<e>` elements, where "e" stands for entry.

```
<clsettings-map>
  <e> … </e>
  <e> … </e>
  <e> … </e>
  …
</clsettings-map>
```

The format of each entry element is as follows:

```
<e>
      <key> … </key>
      <descr> … </descr>
      <pname> … </pname>
      <uprompt> … </uprompt>
      <uterm> … </uterm>
      <pprompt> … </pprompt>
      <pterm> … </pterm>
```

```
        <eprompt> … </eprompt>
        <eterm> … </eterm>
        <cprompt> … </cprompt>
        <cterm> … </cterm>
        <mprompt> … </mprompt>
        <mresp> … </mresp>
        <lgnmenprompt> … </lgnmenprompt>
        <lgnmenresp> … </lgnmenresp>
        <ttype> … </ttype>
    </e>
```

The <key> sub-element defines which devices this entry applies to (in terms of MAC address, IP address, or SNMP OID). The <descr> sub-element provides a textual description of the entry and/or the devices it applies to. The <pname> sub-element specifies which protocol this entry applies to (SSH or Telnet). The other sub-elements define the command-line settings to be used for devices matching this entry.

The <key> sub-element (required)

The <key> sub-element specifies which devices an entry applies to. All devices that match the <key> will use this entry's command-line settings. The <key> may specify the device's MAC address, IP address, or SNMP OID. It works just like the <key> sub-element described previously in the section titled "Custom Device Descriptions, Vendors, and Categories". Please refer to that section for details.

The <descr> sub-element (required)

<descr> stands for "description". The value of this sub-element is a textual description of this entry and/or the devices it applies to.

The <pname> sub-element (required)

<pname> stands for "protocol name". This sub-element specifies which protocol this entry applies to ("ssh" or "telnet").

The <uprompt> sub-element (optional)

<uprompt> stands for "user name prompt". This sub-element specifies the user name prompt displayed by the device during login. Its value can be either a literal string or a regular expression. If this setting is not specified, UVexplorer's default setting will be used, which is:

```
<uprompt>regex:(?i)(username:|user name:|user:|login:|login as:) ?$</uprompt>
```

The <uterm> sub-element (optional)

<uterm> stands for "user name terminator". This sub-element specifies the character sequence sent to the device to terminate the user name. Some devices require CR while others require CRLF. This setting gives you exact control over what characters are sent to the device after the user name. If this setting is not specified, UVexplorer's default setting will be used, which is CR (as shown below):

```
<uterm>&#13;</uterm>
```

The <pprompt> sub-element (optional)

<pprompt> stands for "password prompt". This sub-element specifies the password prompt displayed by the device during login. Its value can be either a literal string or a regular expression. If this setting is not specified, UVexplorer's default setting will be used, which is:

<pprompt>regex:(?i)password: ?$</pprompt>

The <pterm> sub-element (optional)

<pterm> stands for "password terminator". This sub-element specifies the character sequence sent to the device to terminate the password. Some devices require CR while others require CRLF. This setting gives you exact control over what characters are sent to the device after the password. If this setting is not specified, UVexplorer's default setting will be used, which is CR (as shown below):

<pterm>
</pterm>

The <eprompt> sub-element (optional)

<eprompt> stands for "enable prompt". This sub-element specifies the enable prompt displayed by the device when querying the user for the enable password. Its value can be either a literal string or a regular expression. If this setting is not specified, UVexplorer's default setting will be used, which is:

<eprompt>regex:(?i)password: ?$</eprompt>

The <eterm> sub-element (optional)

<eterm> stands for "enable terminator". This sub-element specifies the character sequence sent to the device to terminate the enable password. Some devices require CR while others require CRLF. This setting gives you exact control over what characters are sent to the device after the enable password. If this setting is not specified, UVexplorer's default setting will be used, which is CR (as shown below):

<eterm>
</eterm>

The <cprompt> sub-element (optional)

<cprompt> stands for "command prompt". This sub-element specifies the command prompt displayed by the device when prompting the user for a command to run. Its value can be either a literal string or a regular expression. If this setting is not specified, UVexplorer's default setting will be used, which is:

<cprompt>regex:(?i).*(>|:~|#|]|%|\(enable\))(?$|.*\$?$)</cprompt>

The <cterm> sub-element (optional)

<cterm> stands for "command terminator". This sub-element specifies the character sequence sent to the device to terminate commands. Some devices require CR while others require CRLF. This setting gives you exact control over what characters are sent to the device after commands. If this setting is not specified, UVexplorer's default setting will be used, which is CR (as shown below):

<cterm>
</cterm>

The <mprompt> sub-element (optional)

`<mprompt>` stands for "more prompt". This sub-element specifies the prompt displayed by the device when asking the user to press a key to see the next page of input. Its value can be either a literal string or a regular expression. If this setting is not specified, UVexplorer's default setting will be used, which is:

<mprompt>regex:(?i).*(-+\s*more\s*-+.*|\(.*more.*\).*|more.+<return>|more:.*|press any key to continue:?|press return to continue:?|to continue or <Q> to quit:?) ?$</mprompt>

The <mresp> sub-element (optional)

`<mresp>` stands for "more response". This sub-element specifies the character sequence sent to the device in response to a "more prompt" (i.e., to move to the next page of output). If this setting is not specified, UVexplorer's default setting will be used, which is a space character (as shown below):

<mresp> </mresp>

The <ttype> sub-element (optional)

`<ttype>` stands for "terminal type". This sub-element lets you specify which terminal type UVexplorer should emulate when communicating with the device. Typical choices would be "tty", "std", or "dumb". The goal is to disable terminal control character sequences coming from the device using a terminal type that the device supports. If this setting is not specified, UVexplorer's default setting will be used, which is "tty" (as shown below):

<ttype>tty</ttype>

Custom Device Configuration Command Scripts

UVexplorer can use the SSH and Telnet protocols capture device configurations. UVexplorer captures a device's configuration by logging in to the device, running appropriate commands to display the device's configuration data, capturing the output of those commands, and storing the configuration data in the UVexplorer database.

Many devices distinguish between their "startup configuration" and "running configuration". A device's startup configuration is the persistent configuration the device uses when it boots up. A device's running configuration is the current configuration the device is running with, even if that configuration has not been saved as the startup configuration. Often, the startup and running configurations are the same. However, if an administrator has made configuration changes since last booting the device, but has not saved those changes to the startup configuration, the two configurations will be different. Therefore, UVexplorer captures both startup and running configurations.

The commands required to capture a device's startup and running configurations will vary from device to device. UVexplorer has default built-in scripts that it uses to capture startup and running configurations, and these scripts will work for many devices. However, you may have devices in your IT environment for which the default scripts do not work. In this case, you can configure UVexplorer with custom command scripts that will work for your devices.

Writing Custom Scripts

If you need to specify custom scripts for capturing device configurations, you can do so by creating an XML file with the following name and location:

\ProgramData\UVnetworks\UVexplorer\2.0**user-command-script-map.xml**

The root element of this file should be a <command-script-map> element. Within the root <command-script-map> element you can place any number of <e> elements, where "e" stands for entry.

```
<command-script-map>
  <e> … </e>
  <e> … </e>
  <e> … </e>
  …
</command-script-map>
```

The format of each entry element is as follows:

```
<e>
      <key> … </key>
      <scpttyp> … </scpttyp>
      <descr> … </descr>
      <cmdscpt>
            <cmdtbl>
                  <cmd> … </cmd>
                  <cmd> … </cmd>
                  <cmd> … </cmd>
                  …
            </cmdtbl>
      </cmdscpt>
</e>
```

The <key> sub-element defines which devices this entry applies to (in terms of MAC address, IP address, or SNMP OID). The <scpttyp> sub-element specifies the type of the script, and must be either "BackupStartup" or "BackupRunning". The <descr> sub-element provides a textual description of the script, such as "Cisco Startup Config".

The <key> sub-element (required)

The <key> sub-element specifies which devices an entry applies to. That is, all devices that match the <key> will use this entry's command script. The <key> may specify the device's MAC address, IP address, or SNMP OID. It works just like the <key> sub-element described previously in the section titled "Custom Device Descriptions, Vendors, and Categories". Please refer to that section for details.

The <scpttyp> sub-element (required)

<scpttyp> stands for "script type". The value of this sub-element specifies the type of the script, and must be either "BackupStartup" or "BackupRunning", depending on whether it captures the device's startup or running configuration.

The <descr> sub-element (required)

<descr> stands for "description". The value of this sub-element is a textual description of the script (e.g., "Aruba Running Configuration").

The <cmdscpt> and <cmdtbl> sub-elements (required)

<cmdscpt> stands for "command script". This sub-element contains a <cmdtbl> sub-element, which stands for "command table". The <cmdtbl> sub-element contains a sequence of one or more commands that comprise the command script itself. The commands in the script will be executed sequentially by UVexplorer. Each command is represented as a <cmd> sub-element.

The <cmd> sub-element (required)

Each <cmd> sub-element contains a single command that belongs to the command script. In order to understand how commands work, there are several concepts that must be understood. These preliminary concepts will be discussed first before describing the <cmd> sub-element itself.

Variable Substitutions in Commands

Commands are essentially text strings that are sent to a device for execution. The central attribute of a command is the command string itself (e.g., "show config"). Sometimes it is necessary to include values in a command string that are dynamic. For example, a command might need to reference the device's IP address, which, of course, will be different for each device. In order to support dynamic values in command strings, UVexplorer supports the notion of "variable substitution". When you write a command string, you can reference variables whose values are automatically defined by UVexplorer. The values of these variables can be referenced using a dollar-sign. For example, this command references the IP address of the current device on which the script is being executed:

```
<cmd>
      <cmdtype>Execute</cmdtype>
      <val>set ip-address $(Device.IpAddress)</val>
      …
</cmd>
```

Notice that the syntax used to reference a variable is a dollar-sign followed by the variable name wrapped in parentheses. Before sending the command string to the device for execution, UVexplorer will replace all variable references with their values. The following table shows all the variables that can be referenced in command strings.

Variable Name	Variable Value
Device.IpAddress	The device's primary IP address
Device.SystemName	The device's SNMP system name value
Device.SystemContact	The device's SNMP system contact value
Settings.UserName	The user name value in the SSH/Telnet settings
Settings.Password	The password value in the SSH/Telnet settings
Settings.EnablePassword	The enable password in the SSH/Telnet settings
Settings.UserNamePrompt	The user name prompt value in the command-line settings
Settings.UserNameTerminator	The user name terminator value in the command-line settings
Settings.PasswordPrompt	The password prompt value in the command-line settings
Settings.PasswordTerminator	The password terminator value in the command-line settings
Settings.EnablePrompt	The enable prompt value in the command-line settings
Settings.EnableTerminator	The enable terminator value in the command-line settings
Settings.CommandPrompt	The command prompt value in the command-line settings
Settings.CommandTerminator	The command terminator value in the command-line settings

System.MorePrompt	The more prompt value in the command-line settings
System.MoreResponse	The more response value in the command-line settings

Since the dollar-sign syntax is used to reference variables in command strings, if you need to include a literal dollar-sign character in a command string, you must represent it as $$. When two consecutive dollar-signs are found in a command string, it is converted to a single literal dollar-sign and not treated as a variable reference.

Storing Command Outputs in the UVexplorer Database

The primary intent of command scripts is to capture the startup and/or running configurations of devices, and to store those configurations in the UVexplorer database. A device's startup and running configurations are stored in the "Config" section of the device to which they belong (a device's "Config" can be viewed in the UVexplorer user interface). A device's "Config" contains key/value pairs, where each pair stores a captured device configuration under a certain key. Startup configurations are stored under the key "startup-config", and running configurations are stored under the key "running-config".

When you write a custom command script to capture device configurations, if you want a command's output to be stored in the device's "Config", you should specify a "capture key" for that command. The "capture key" tells UVexplorer what key to store the command's output under in the device's "Config". For example, this command uses the capture key "startup-config":

```
<cmd>
      <cmdtype>Execute</cmdtype>
      <val>show start</val>
      <cptrkey>startup-config</cptrkey>
      …
</cmd>
```

In general, you should use the standard "startup-config" and "running-config" capture keys when writing your custom scripts. However, this is not enforced, and you can actually use whatever capture keys you want to store command outputs in a device's "Config". After running your scripts, you will see the capture keys you used in your script appear in the device's "Config".

NOTE: Your scripts may contain commands that have no output, or commands that have output you have no interest in capturing. You should not give such commands capture keys in your script so that their outputs will not be stored by UVexplorer.

Command Output Editing

When capturing command outputs, it is sometimes desirable to edit a command's output before it is stored in the UVexplorer database. For example, many command outputs contain extraneous lines of text either before or after the interesting part of the output. In this case it may be desirable to "trim" (or remove) the extraneous lines before the output is stored. For situations like this, as part of a command's definition you can specify editing operations that should be performed on the command's output before it is stored in the database. For example, suppose the "show start" command's output has several blank lines before the first meaningful line, and that the first meaningful line starts with an exclamation point (!). To remove the blank lines from the beginning of the output, you could tell UVexplorer to "Trim Before !", as shown in this command:

```
<cmd>
      <cmdtype>Execute</cmdtype>
      <val>show start</val>
      <cptrkey>startup-config</cptrkey>
      <trmeditors>
            <trmeditor>
                  <trimtyp>TrimBefore</trimtyp>
                  <ptrn>!</ptrn>
                  <isrgx>false</isrgx>
            </trmeditor>

      </trmeditors>
</cmd>
```

This tells UVexplorer to throw away all command output before the first '!'.

In general, a command may contain a <trmeditors> sub-element. <trmeditors> stands for "trim editors". This sub-element contains a sequence of <trmeditor> sub-elements, each of which defines an editing operation that UVexplorer should perform on the command's output. The following editing operations are available:

1. Trim Start Lines: Remove the first N lines from the command's output. <trimtyp> stands for "trim type". <lncnt> stands for line count, which specifies the number of lines to be removed.

   ```
   <trmeditor>
         <trimtyp>TrimStartLines</trimtyp>
         <lncnt>2</lncnt>
   </trmeditor>
   ```

2. Trim End Lines: Remove the last N lines from the command's output. <lncnt> specifies the number of lines to be removed.

   ```
   <trmeditor>
         <trimtyp>TrimEndLines</trimtyp>
         <lncnt>2</lncnt>
   </trmeditor>
   ```

3. Trim Start: Remove all output before and including the first occurrence of a specified literal string or regular expression.

   ```
   <trmeditor>
         <trimtyp>TrimStart</trimtyp>
         <ptrn>#&#13;#&#13;#&#13;</ptrn>
         <isrgx>false</isrgx>
   </trmeditor>
   ```

4. Trim End: Remove all output including and after the last occurrence of a specified literal string or regular expression.

```
<trmeditor>
      <trimtyp>TrimEnd</trimtyp>
      <ptrn>#&#13;#&#13;#&#13;</ptrn>
      <isrgx>false</isrgx>
</trmeditor>
```

5. Trim Before: Remove all output before the first occurrence of a specified literal string or regular expression.

```
<trmeditor>
      <trimtyp>TrimBefore</trimtyp>
      <ptrn>!</ptrn>
      <isrgx>false</isrgx>
</trmeditor>
```

6. Trim After: Remove all output after the last occurrence of a specified literal string or regular expression.

```
<trmeditor>
      <trimtyp>TrimAfter</trimtyp>
      <ptrn>!</ptrn>
      <isrgx>false</isrgx>
</trmeditor>
```

7. Remove Lines: Remove all lines that match a specified literal string or regular expression.

```
<trmeditor>
      <trimtyp>RemoveLines</trimtyp>
      <ptrn>system time.+</ptrn>
      <isrgx>true</isrgx>
</trmeditor>
```

Ignoring Commands That Fail

When UVexplorer runs a script on a device, it runs each command in the script sequentially one-by-one. If a command fails to execute, UVexplorer immediately aborts the script execution, and reports an error. Occasionally, it is useful to have UVexplorer continue running a script even if a command fails. For this reason, when defining a script command, you can tell UVexplorer to ignore the command's result, and keep running the script even if that command fails to execute. The most common use of this feature is on commands that terminate the interactive session with the device, such as "logout", "exit", or "quit". Such commands usually terminate the connection with the device, and will thus fail to execute from UVexplorer's perspective. For such a command, you can tell UVexplorer to "ignore the result" of the

command, and not treat its failure to execute as an error. The follow example command shows how this is done:

```
<cmd>
      <cmdtype>Execute</cmdtype>
      <val>exit</val>
      <ignoreresult>true</ignoreresult>
</cmd>
```

High-level Commands and Low-level Commands

UVexplorer command scripts can contain both "high-level" and "low-level" commands. High-level commands are commands that try to do a lot of the work for you, relieving you of the burden of specifying every small detail of a device interaction. The most common example is logging in to a device. The process of logging in to a device requires the following steps:

1. Wait until you see the user name prompt.
2. Send the user name to the device followed by the user name terminator.
3. Wait until you see the password prompt.
4. Send the password to the device followed by the password terminator.
5. Wait until you see a command prompt.
6. Now you're logged in.

This login process can actually be more complicated on some devices, such as devices that require the selection of a menu option before you can log in, or devices that produce paged output before letting you log in. As you can see, the process of logging in is fairly involved. For this reason, UVexplorer has a high-level command named "Login" that will do all of the login steps for you automatically.

UVexplorer supports three different high-level commands:

1. Login: log in to the device
2. Enable: raise the script to a higher privilege level by entering the enable password
3. Execute: execute a specified command string, capture its output if requested, and handle paged output by entering responses to more prompts

Most scripts can be written solely in terms of these three high-level commands. However, if UVexplorer's high-level commands don't work for one of your devices, you can fall back and use UVexplorer's low-level commands, which can do virtually anything (although you have to work harder at it). The low-level commands are:

1. Read: receive output from the device, and send input to the device depending on what output was received.
2. Write: send a specified input to the device.

You can also combine high-level and low-level commands in the same script in order to accomplish your goals. The following sections provide more detail on each of the high and low-level commands.

High-level Script Commands

The "Login" Command

The Login command executes the process of logging in to the device. It is specified like this:

```
<cmd>
      <cmdtype>Login</cmdtype>
</cmd>
```

When the Login command executes, it connects to the device, and inspects the output from the device. If the device output contains the user name prompt, UVexplorer sends the user name and user name terminator to the device. If the device output contains the password prompt, UVexplorer sends the password and password terminator to the device. If the device output contains the more prompt, UVexplorer sends the more response to the device. If the device output contains the login menu prompt, UVexplorer sends the login menu response to the device. If the device output contains the command prompt, UVexplorer assumes that the login process is complete, and the Login command terminates successfully. If the device output contains none of the above, then the Login command fails.

The "Enable" Command

The Enable command raises the privilege level of the script by entering the enable password. It is specified like this:

```
<cmd>
      <cmdtype>Enable</cmdtype>
</cmd>
```

When the Enable command executes, it inspects the output from the device. If the device output contains the enable prompt, UVexplorer sends the enable password and enable terminator to the device. If the device output contains the command prompt, UVexplorer assumes that the enable process is complete, and the Enable command terminates successfully. If the device output contains none of the above, then the Enable command fails.

The "Execute" Command

The Execute command sends a specified command string to the device for execution. It is specified like this:

```
<cmd>
      <cmdtype>Execute</cmdtype>
      <val>show start</val>
      <cptrkey>startup-config</cptrkey>
      <trmeditors>
            <trmeditor>
                  <trimtyp>TrimBefore</trimtyp>
                  <ptrn>!</ptrn>
            </trmeditor>
            <trmeditor>
                  <trimtyp>TrimEndLines</trimtyp>
                  <lncnt>2</lncnt>
            </trmeditor>
      </trmeditors>
      <ignorereslt>false</ignoreresult>
</cmd>
```

The <val> sub-element contains the command string to be sent to the device for execution. This is required.

The <cptrkey> sub-element contains the "capture" key under which the command's output will be stored in the device's "Config". This is optional.

The <trmeditors> sub-element contains the editing operations to be performed on the command's output before storing it in the database. The syntax of this sub-element was described previously in the section titled "Command Output Editing". Please refer that section for details. This is optional.

The <ignoreresult> sub-element specifies whether UVexplorer should continue executing the script if this command fails (true means continue execution, false means stop execution). This is optional. The default value is "false".

When a command of type "Execute" runs, UVexplorer does the following:

1. Performs variable substitution on the command string (i.e., replaces all referenced variables with their values).
2. Sends the command string to the device followed by the command terminator.
3. Reads the output sent back by the device.
4. If the device output matches the enable prompt, this means that the enable password must be entered before the command can be executed. Therefore, UVexplorer sends the enable password and the enable terminator to the device.
5. If the device output ends with the more prompt, this means that the command output is paged. Therefore, UVexplorer sends the more response to the device.
6. Whether the command output is paged or not, UVexplorer captures the command output.
7. When the device output ends with the command prompt, the command output is complete.
8. If the command specifies any trim editors, the trim editors are applied to the captured command output.
9. If the command specifies a capture key, the trimmed command output is stored in the device's "Config" under the specified capture key.

Low-level Script Commands

The high-level Login, Enable, and Execute commands are typically sufficient to write a custom command script. However, in some cases they might not work with a device with non-standard behavior. In this case, you will need to drop down and use the low-level script commands to write your script. There are only two low-level commands: Write and Read. If you think about it, the only two operations you can really perform on a device over SSH or Telnet are sending input to the device and reading output from the device. Therefore, the low-level Write and Read commands can be used to do just about anything to a device through its SSH or Telnet interface.

The "Write" Command

The Write command writes a specified character string to the device. The Write command's syntax is as follows:

```
<cmd>
      <cmdtype>Write</cmdtype>
      <val>echo $(Device.MacAddress)</val>
```

```
</cmd>
```

This is how your script provides input to the device. The Write command can be used to perform the following tasks:

1. Send a user name and user name terminator to the device.
2. Send a password and password terminator to the device.
3. Send an enable password and enable terminator to the device.
4. Send a more response to the device.
5. Select a menu option from a menu displayed by the device.
6. Send a command string and command terminator to the device.
7. Send whatever other input that is required by the device's SSH/Telnet interface.

When a Write command executes, UVexplorer does the following:

1. Performs variable substitution on the `<val>` string.
2. Sends the resulting string to the device.
3. If a command echo is received back from the device, throw it away. (Devices often echo command inputs back to the client so the client can see the command they sent. For our purposes this command echo is not useful, and UVexplorer discards it.)

The "Read" Command

The Read command reads output from the device, and, depending on the output received, optionally sends an input back to the device.

The simplest form of the Read command looks like this:

```
<cmd>
      <cmdtype>Read</cmdtype>
      <val>User Name: </val>
</cmd>
```

When this command executes, UVexplorer does the following:

1. Performs variable substitution on the `<val>` string.
2. Reads output from the device.
3. If the received output matches the variable-substituted `<val>` string, the command succeeds. Otherwise, the command fails.

The complex form of the Read command is demonstrated in the command sequence below. This command sequence uses log-level commands to log in to a device and capture the device's startup configuration.

```
<cmd>
  <cmdtype>Read</cmdtype>
  <val>$(Settings.UserNamePrompt)</val>
</cmd>
<cmd>
  <cmdtype>Write</cmdtype>
  <val>$(Settings.UserName)&#13;</val>
```

```
</cmd>
<cmd>
  <cmdtype>Read</cmdtype>
  <val>$(Settings.PasswordPrompt)</val>
</cmd>
<cmd>
  <cmdtype>Write</cmdtype>
  <val>$(Settings.Password)&#13;</val>
</cmd>
<cmd>
  <cmdtype>Read</cmdtype>
  <val>$(Settings.CommandPrompt)</val>
</cmd>
<cmd>
  <cmdtype>Write</cmdtype>
  <val>show start&#13;</val>
</cmd>
<cmd>
  <cmdtype>Read</cmdtype>
  <cptrkey>startup-config</cptrkey>
  <xpctrules>
    <xpctrule>
      <prompt>$(Settings.MorePrompt)</prompt>
      <response>$(Settings.MoreResponse)</response>
      <append>true</append>
    </xpctrule>
    <xpctrule>
      <prompt>$(Settings.CommandPrompt)</prompt>
      <append>true</append>
    </xpctrule>
  </xpctrules>
</cmd>
```

The heart of a complex Read command is the <xpctrules> sub-element. <xpctrules> stands for "expect rules". This sub-element contains a sequence of <xpctrule> sub-elements, each of which represents an "if-then" rule: If the output received from the device matches the rule's <prompt>, send the rule's <response> to the device. The rule's <append> sub-element controls whether or not the device output that matched the rule's prompt should be included in the Read command's output. The sequence of expect rules acts much like a "switch" or "case" statement in programming languages: Read the device's output, find the rule that matches the output, and send the rule's response to the device.

When UVexplorer executes a complex Read command, it does the following:

1. Performs variable substitution on the <prompt> and <response> of each expect rule.
2. Reads output from the device.
3. Searches for an expect rule whose prompt matches the received output.
4. If no expect rule matches the received output, the Read command fails.
5. If an expect rule matches the received output,

   a. If the matching expect rule has a “true” `<append>` value, append the received output to the Read command’s output.
   b. If the matching expect rule has a `<response>`, send the rule’s response to the device, and go back to step 2.
   c. If the matching expect rule does not have a `<response>`, the Read command terminates successfully.
6. If the Read command specifies any trim editors, the trim editors are applied to the Read command’s output.
7. If the Read command specifies a capture key, the trimmed command output is stored in the device’s “Config” under the specified capture key.