Unsniff Scripting Guide

For use with Unsniff Network Analyzer

Version 1.2
Feb 18, 2006

Version 1.2 Feb 18, 2006

Unleash Networks Pvt Limited
5, Nehru Street, Gowrivakkam
Chennai 601302, India
http://www.unleashnetworks.com
support@unleashnetworks.com

Version 1.2 Feb 18, 2006

Revision History

June 20 , 2005	Rev 1.0	Initial Release
Sep 26, 2005	Rev 1.1	Updated prior to Beta Release
Feb 18, 2006	Rev 1.2	Comments from Beta 1, documented FindField for sub fields, documented OpenForRead / OpenForWrite methods

1 Introduction

This guide explains how you can use the scripting features of Unsniff to write your own powerful analysis tools.

1.1 About Unsniff Scripting

Unsniff is the next generation network analyzer software from Unleash Networks. It features never before seen graphical representations of packets,PDU analysis, full stream analysis,a new storage format that can store entire sessions, PDUs, User Objects, annotations,and more. All conventional features of a network/protocol analyzer such as filters, statistics are also present in Unsniff in an improved form.

The two features that really set Unsniff apart from the other tools are:

1. **Scriptability** : Enables you to write your own powerful analysis scripts
2. **Extensibility** : Extend Unsniff by adding user interface elements or custom protocol decoders.

This document addresses the *scriptability* features of Unsniff. If you are interested in writing custom decoders (or) extending the Unsniff user interface – refer to the "*Unsniff API Developers Guide*".

Many network analysts are talented professionals who regularly write their own tools using scripting languages like Perl, Shell, or VBScript. Unsniff is the first network analyzer that enables network analyzers to write their own scripts for performing custom tasks. From monitoring digital certificates to checking for network performance problems – you can do it all via the Unsniff Scripting API.

1.2 Intended audience

This document is intended for developers who want to:

- Write Scripts to perform custom tasks on captured data

1.2.1 Skills required

You need to be familiar with at least one scripting language to use the Unsniff Scripting API. The standard scripting language on Windows Platforms is "Visual Basic Scripting Edition" also known as VBScript. We recommend the **Ruby** scripting language. Its object oriented design and concise but easy-to-maintain structure make it ideal to build complex, reusable network analysis scripts. The Fox-Ruby toolkit allows you to write user interfaces for your scripts easily. *The examples in this guide and the sample code provided in the API are in Ruby and VBScript.*

You can also use Perl, Jscript, and Python or any other scripting language that provides access to Windows Automation Objects.

1.3 Getting Started

You need the following resources to effectively use the Scripting API

1. A licensed copy of ***Unsniff Network Analyzer***
   - You can purchase a licensed copy or download a trial from http://www.unleashnetworks.com
2. If you are planning to use VBScript, it is pre-installed on all Windows 2000 or XP systems. You do not need to download it separately
3. If you are planning to use Ruby, download ruby from http://www.rubylang.org
4. If you are planning to use Perl/Python/others. Install the respective runtimes on the system
5. You can find a whole lot of scripting resources online at the "*Unsniff DevZone"* http://www.unleashnetworks.com

Info
Unleash Networks maintains an online script library at http://www.unleashnetworks.com/script-contents.html . This library contains many scripts written by users of Unsniff that can be downloaded and used for free. You may also share your most useful scripts with others by posting on our website.

1.3.1 Platforms

The Unsniff Plugin API works only on the following platforms:

- Windows 2000
- Windows XP

2 Script variations

Unsniff supports two types of scripts based on how they interact with Unsniff Network Analyzer.

1. **Stand alone scripts**
   These scripts typically are run from the command line or via a Windows shortcut. They operate on capture files outside the Unsniff application.

2. **Integrated scripts**
   You can attach custom scripts to many popup menu items in Unsniff. They are triggered when the corresponding menu item is selected. These scripts give you access to the currently open capture file and the current selection context. This is a powerful way to add functionality to the Unsniff application.

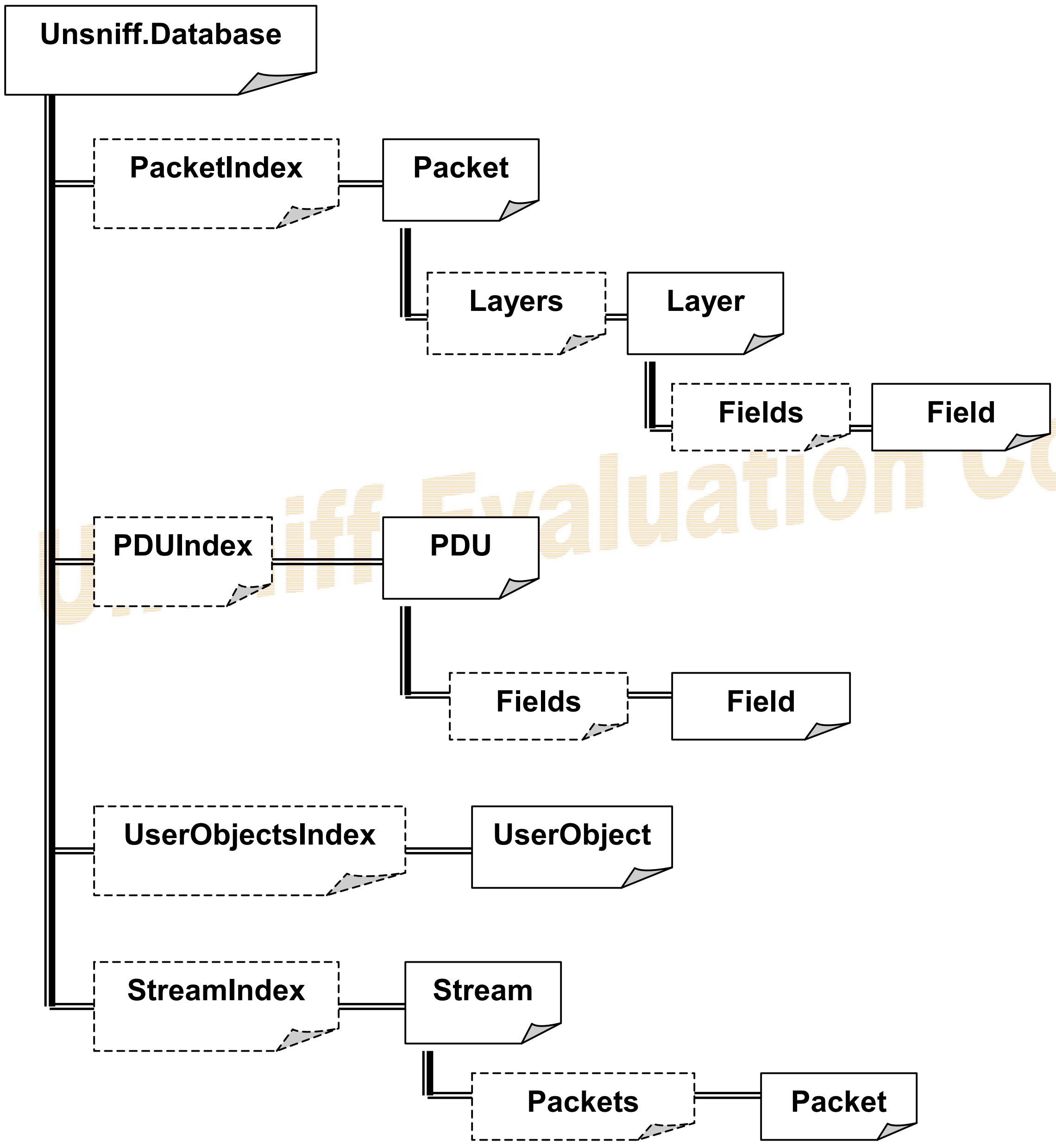
Both types of scripts use the same object model. It is fairly easy to write scripts that can work in both standalone mode and in integrated mode. The integrated mode gives you access to the currently open capture file and various selection contexts.

3 *Scripting Object Model*

The scripting interface consists of a single top-level object called "`Unsniff.Database`". This object represents a single capture file stored in the Unsniff (**.usnf)* format. Your task is to get hold of the "Unsniff.Database" object and work your way through the other objects. If you are writing an integrated script – you can additionally access the currently open document, various selection contexts, and the scripting console. See Section 4 for more details on integrated scripts.

3.1 Object Model Diagram

The following picture shows how the Object Model is organized. Only object names and their relationships are shown here.



3.2 Object Creation

The Unsniff Scripting Object Model is a hierarchical structure. Only one top-level object *"Unsniff.Database* is publicly creatable via the Prog ID "Unsniff.Database". All other objects are accessed via methods and properties of already created objects.

To create the root object:

VBScript

```
Set MyDB = CreateObject ("Unsniff.Database")
```

Ruby

```
MyDB = Win32OLE.new ("Unsniff.Database")
```

3.2.1 A simple example

Let us consider a simple example. In this example, we will write a script to print the description of each packet in a given Unsniff capture file. This example will illustrate the following concepts.

- The structure of a typical script application
- How the root object is created and accessed
- How you can navigate to the other objects

Example: Print the description of each packet in a given capture file.

VBScript

```
' -----------------------
' Check usage & arguments
' -----------------------
if WScript.Arguments.Count <> 1 then
      WScript.Echo "Usage: prpidx <filename>"
      WScript.Quit
end if

ArgFile  = WScript.Arguments.Item(0)

' ----------------------------------------
' Open the file & navigate to packet index
' ----------------------------------------
Set UnsniffDB = CreateObject("Unsniff.Database")
UnsniffDB.Open(ArgFile)

Set PacketStore = UnsniffDB.PacketIndex

For Each Packet In PacketStore
      WScript.Echo     Packet.Description
Next

UnsniffDB.Close()
```

Ruby

```
require 'win32ole'

USAGE = "prpidx <capture-filename>"

#
# function: print the description
#
def printPacket(packet)
       $stdout << packet.Description << "\n"
end

#
# check arguments
#
if ARGV.length != 1
       puts USAGE
       exit 1
end

UnsniffDB = WIN32OLE.new("Unsniff.Database")
UnsniffDB.Open(ARGV[0])
Count = UnsniffDB.PacketCount

PacketStore = UnsniffDB['PacketIndex']

(0..Count-1).each{ |idx| printPacket(PacketStore.Item(idx)) }

UnsniffDB.Close()
```

3.3 Objects Reference

This section describes the properties and methods supported by each object in the Unsniff Scripting Object Model.

3.3.1 Unsniff.Database

Description
Represents a capture file. You must first create this object and then use this to open an existing capture file or create a new capture file. You can then use the methods and properties provided by this object to navigate to other interesting parts of the capture file. The Unsniff.Database is the only object that can be publicly created via its ProgID.

Properties

Name	Type	Access	Description
PacketCount	Long	Read	The number of packets currently present in the capture database
PDUCount	Long	Read	The number of PDUs currently present in the capture database
PacketIndex	Collection	Read	A collection of Packet objects
PDUIndex	Collection	Read	A collection of PDUs
StreamIndex	Collection	Read	A collection of Streams. *Each stream represents a TCP/IP session.*
UserObjectsIndex	Collection	Read	A collection of User Objects. *Examples of user objects are images, HTML, audio, RTP media, files, etc*

Methods

Name	Parameters	Description
Open	Filename (*String*)	Opens the capture file identified by the filename parameter for read-write access. The filename can be a full pathname or a relative filename. You can also open the file explicitly for readonly or readwrite access using the *OpenForRead* and *OpenForWrite* methods.
OpenForRead	Filename (*String*)	Opens the capture file identified by the filename for read only. Use this method if you are just analyzing a capture file and not trying to change its contents.
OpenForWrite	Filename (*String*)	Open the capture file for read-write access. Use this method if you want to change the contents of the capture file in any way. ⓘ Info: This call will return an error if the capture file is already opened in the main Unsniff application. Try OpenForRead or close the capture file in Unsniff while your script is running.

New	Filename (*String*)	Creates a new capture file with the given filename. The filename can be a full pathname or a relative filename.
Close	None	Close the file. The file must be currently open via the *Open* or *New* methods. All changes made to a file opened with write access are saved.
BeginExport	Filename (*String*) Type (*String*)	Open an export file with the given name and type. Currently the only type supported is “libpcap”. *To export an entire file:* Use the Export() method *To selectively export packets:* Call BeginExport(), followed by a bunch of ExportXXX() calls, then with an EndExport()
ExportPacket	Packet	Export this packet to the export file currently opened via BeginExport()
ExportStream	Stream	Export the entire stream (*e.g. TCP/IP session*) to the export file currently opened via BeginExport()
EndExport	None	Close the export file previously opened via BeginExport()
Export	Type(*String*) Filename (*String*)	Export all the packets in this capture file to the given file. The desired export format is specified in the Type parameter. Currently the only Type supported is “libpcap”
Import	Type(*String*) Filename (*String*)	Import all the packets in a capture file in another format into this file. Currently - Type must be set to “libpcap”
AddPacket	Packet	Add the given Packet to this capture file. This packet could be from another capture file that is currently open.
AddStream	Stream	Add the given Stream to this capture file. The stream could be from another capture file that is currently open.

3.3.2 Collection Objects

Description
A collection is used to conceptually store a group of objects of the same type. You can use standard scripting methods to access the contents of a collection.

Properties

Name	Type	Access	Description
Count	Long	Read	The number of objects stored in this collection

Methods

Name	Parameters	Description
Item	Long	Returns the Item at this index. The items are zero-indexed. This method is implicitly called if you use the array operators in most scripting languages. For example: PacketStore(10) is internally translated to PacketStore.Item(10).

Usage Notes
You can use the For..Next or the For Each method to iterate through a collection. Consult your scripting language for the corresponding methods. VBScript and Ruby Examples are shown below.

VBScript

```
' Use the For Each statement
Set PacketStore = UnsniffDB.PacketIndex
For Each Packet In PacketStore
      WScript.Echo     Packet.Description
Next

' Use the For statement
Set PacketStore = UnsniffDB.PacketIndex
NumPackets = PacketStore.Count
For I = 0 To NumPackets-1
      Set Packet = PacketStore(I)
      WScript.Echo     Packet.Description
Next
```

Ruby
At Unleash Networks; Ruby is our favorite scripting language. The following examples illustrate how collections are accessed in Ruby. It does not get any terser and easy to maintain than this.

```
' Use the Count
PacketStore = UnsniffDB['PacketIndex']
Count = PacketStore['Count']
(0..Count-1).each{ |idx| print PacketStore.Item(idx).Description }

' Use the each block
Set PacketStore = UnsniffDB.PacketIndex
PacketStore.each { |packet| print packet.Description }
```

3.3.3 Packet

Description
Represents a single packet present in the capture file.

Properties

Name	Type	Access	Description
ID	Long	Read	A unique ID for each packet assigned by Unsniff
Description	String	Read/Write	A text description of the packet. This is the description that appears in the packet index list in Unsniff
Type	String	Read	The type of packet. In most cases this is the name of the highest layer protocol present in the packet.
Length	Long	Read	Length of the packet. This is the number of bytes captured by Unsniff. If you have specified a smaller capture length in Unsniff , the actual size of the packet on the wire may be more. See WireLength.
WireLength	Long	Read	The length of the packet on the wire. In most cases this will be equal to the *Length* property. If the packet was truncated the *WireLength* will be greater than *Length*
IsBookmarked	BOOL	Read/Write	You can use this to check if a packet is bookmarked or to set/clear a bookmark
IsAnnotated	BOOL	Read/Write	Annotations are small notes attached to a packet by a network analysis professional. This aids in packet analysis when these files are accessed later. Use this property to check if an annotation exists or to set/clear an annotation
Annotation	String	Read/Write	Use this to query or to set an annotation.
Timestamp	String	Read	A string representation of the timestamp of the packet. The format of this timestamp is determined from your current Windows Locale settings.
TimestampSecs	Long	Read	The seconds part of the packet timestamp. This number returns the number of seconds since midnight January 1, 1900
TimestampUSecs	Long	Read	The microseconds part of the packet timestamp.
Timestamp	String	Read	A string representation of the timestamp of the packet. The format of this timestamp is determined from your current Windows Locale settings.
SourceAddress	String	Read	The source address of this packet. The destination address of this packet. If the

			address has been resolved to a name – this property contains the source name.
DestinationAddress	String	Read	The destination address of this packet. If the address has been resolved to a name – this property contains the destination name.
Layers	Collection	Read	Get all the layers contained in this packet. You have to first access the layer object to get at the individual fields of a packet.

Methods

Name	Parameters	Description
FindLayer	LayerName(*String*)	Find a protocol layer within this packet. *Example*: `Set UDPLayer = Packet.FindLayer("UDP")`
FindLayerByGUID	LayerGUID(*String*)	Find a protocol layer within this packet with the specified GUID. Use this version for higher performance than finding layer by name *Example*: `Set UDPLayer = Packet.FindLayer("{14D7AB53-CC51-47e9-8814-9C06AAE60189}")`
WireLength	Long	Actual length of the packet. If you have specified a smaller capture length in Unsniff , the number of bytes captured could be less than the WireLength.
RawData	String	A hex dump of the entire packet data. You must interpret the hex within your captue file.

3.3.4 Layer

Description
The layer object represents a protocol layer within a packet. For example an HTTP packet may have "Ethernet" , "IP", "TCP", "HTTP" layers. These are modeled using the Layer object

Properties

Name	Type	Access	Description
Name	String	Read	The name of the layer. This is usually the short name of the protocol.
ProtID	String	Read	The GUID of the protocol layer. The GUID is returned as a string in Registry format. *You may recall that in the Unsniff plugin architecture each protocol must be assigned a unique GUID.*
Size	Long	Read	The number of bytes in this layer.
Fields	Collection	Read	Get all the fields contained in this layer. This is a collection. For example: In the Ethernet layer: you may have the "Dest MAC,"Src MAC", "Ethertype" fields. This is your main method to drilldown to field level details from a packet.

Methods

Name	Parameters	Description

FindField	FieldName (*String*)	Find a field in this layer using a field name.
		The field name must be as it appears in Unsniff. This method returns the first field that matches the name. All sub fields are searched for a match.
		Example:
		`Set IpSrc = iplayer.FindField("Src Address")`
		
		This method also allows you to specifically search fields within records using a special notation. Notation: ">Group 1>Sub Group2> MyField". There is no limit on the number of groups that can be nested this way. When you use this notation, FindField will search "*Group 1*" for a field named "*Sub Group2*", then search "*Sub Group 2"* for a field named "*MyField*". Use this method to disambiguate duplicate field names or to cut down on exhaustive searches.
		
		Example:
		`Set Fbit = iax.FindField(">FULL FRAME>Source Call Number>F")`

RawData	String	A hex dump of bytes in this layer only.

3.3.5 Field

Description
Represents a single protocol field. If this field is a record or a group field then this field contains other nested fields (called subfields).

Properties

Name	Type	Access	Description
Name	String	Read	The name of the field. This is usually the full name of the field
Value	String	Read	The value of the field as a string. Most scripting language support dynamically converting strings to numbers if you want to work with integer field values
SizeBits	Long	Read	The size of the field in bits
OffsetBits	Long	Read	The offset of the field (in bits) starting from the beginning of this layer.
FieldID	Long	Read	Each field is assigned a unique ID by the plugin developer. You may find this ID useful if you are a plugin developer.
HelpID	Long	Read	A help ID is assigned to each field. This provides bubble help for each field.
SubFieldCount	Long	Read	The number of sub-fields contained in this field. Some fields such as Flags, Records, Sequences may contain other fields. The SubFieldCount property returns the number of immediate subfields. If the SubFieldCount is 0, you can be sure that this is a simple field and does not contain any sub-fields.
SubFields	Collection	Read	If this field has subfields, this collection object contains a list of such sub fields. Each object in the collection is again a Field object
RawData	String	Read	A hex dump of this fields data. You can use this to format your own field values.

Methods

This object does not define any methods

3.3.6 PDU

Description
A Protocol Data Unit is a block of data that is independent of packet boundaries. For stream based protocols the PDU is the meaningful unit of data. PDUs do not respect packet boundaries at all. Unsniff tracks all PDUs in addition to packets, this allows for powerful stream based protocol analysis capabilities. The PDU object represents a single PDU present in the capture file.

Properties

Name	Type	Access	Description
ID	Long	Read	Each PDU is assigned a unique ID by Unsniff
ProtID	String	Read	The Protocol GUID of the PDU. Each protocol in Unsniff must have a unique GUID. The string returned in a GUID in the registry format
Name	String	Read	The Name of the PDU. In most cases, this is the protocol name of the PDU.
Description	String	Read/Write	The text description of the PDU. Your script can also change the description based on your analysis.
SenderAddress	String	Read	The network address of the Sender of this PDU. This is a network name if this address has been resolved to a name.
ReceiverAddress	String	Read	The network address of the Receiver of this PDU. This is a network name if this address has been resolved to a name.
Timestamp	String	Read	The time this PDU was created. The time is returned in a string. The format of the time is determined by the current Windows Locale settings
TimestampSecs	Long	Read	The seconds' part of the PDU create timestamp. This number returns the number of seconds since midnight January 1, 1900
TimestampUSecs	Long	Read	The microseconds part of the PDU create timestamp.
Length	Long	Read	The length (in bytes) of this PDU
Fields	Collection	Read	This collection object contains all the fields in the PDU.
RawData	String	Read	A hex dump of this PDU.

Methods

This object does not define any methods

3.3.7 Stream

Description
This object represents a complete TCP/IP session[1]. Unsniff allows you to work with complete TCP/IP sessions while performing post-capture analysis. You can write simple scripts to perform complex tasks that would be impossible or painfully difficult with other legacy network analyzers.

Some examples[2]:

- Print a list of all TCP/IP sessions that transferred more than 2M bytes total
- Export the top 5 busiest TCP/IP sessions to a libpcap file
- Reassemble and save the first 100 bytes of each TCP session

Properties

Name	Type	Access	Description
ID	Long	Read	Each stream is assigned a unique ID by Unsniff
InSegmentCount	Long	Read	Number of segments from Destination to Source. *For TCP the InSegmentCount is the number of segments in the opposite direction of the initial SYN packet*
OutInSegmentCount	Long	Read	Number of segments from Source to Destination. *For TCP the OutSegmentCount is the number of segments in the same direction of the initial SYN packet*
InByteCount	Long	Read	Number of bytes from Destination to Source. *(in the opposite direction to the initial SYN packet)*
OutByteCount	Long	Read	Number of bytes from Source to Destination *(in the same direction as the initial SYN packet)*
StartTimeStamp	String	Read	The timestamp when the session started. For TCP, this is when the first SYN packet was seen. The format of the timestamp string is determined by the Windows Locale settings
EndTimeStamp	String	Read	The timestamp when the session ended. The session can end due to the normal FIN sequence or RST or due to user stopping the capture prematurely. For format of the timestamp string is determined by the Windows Locale settings
StartTimestampSecs	Long	Read	The seconds part of the *start* timestamp. This number returns the number of seconds since midnight January 1, 1900
StartTimestampUSecs	Long	Read	The microseconds part of the *start* timestamp.

[1] Future versions of Unsniff will support other types of streams in addition to TCP/IP
[2] Think about how difficult these tasks would be to accomplish using your old network analyzer

EndTimestampSecs	Long	Read	The seconds part of the *end* timestamp. This number returns the number of seconds since midnight January 1, 1900
EndTimestampUSecs	Long	Read	The microseconds part of the *end* timestamp.
Description	String	Read/Write	The text description of the stream. Your script can also change the description based on your analysis.
SourceAddress	String	Read	The network address of the source of this stream. A network name is returned if this address has been resolved to a name. For TCP, the source is the station that sent the initial SYN segment.
DestinationAddress	String	Read	The network address of the destination of this stream. A network name is returned if this address has been resolved to a name. For TCP, the source is the station that sent the SYN+ACK response to the initial SYN segment.
Packets	Collection	Read	All the packets that make up this stream. This includes error packets, for example late arrivals, duplicate packets, out of order packets, etc. If you want to perform custom stream analysis you may want access to these packets

Methods

Name	Parameters	Description
SaveToFile	FileName (*String*) Direction (*String*) SeekPos (*Long*) NumBytes (*Long*)	Reassemble and save the contents of this stream. You can save either direction beginning at any offset and any number of bytes. *FileName*: Can be a pathname or a relative filename *Direction* : "**in**" for incoming; "**out**" for outgoing *SeekPos*: 0 for beginning of stream *NumBytes*: Number of bytes to write

3.3.8 UserObject

Description
At the top of the Unsniff food chain is the user object. This can be anything that is of great interest to the network analysis professional. You can write plugins to extract any type of user object from observed traffic. Using the Unsniff Scripting API you can automate all aspects of user objects.

Some examples:
- Save all images greater than 75K in size to a directory
- Export all RTP audio conversations from a given SIP Phone to a directory

Properties

Name	Type	Access	Description
ID	Long	Read	Each user object is assigned a unique ID by Unsniff
IID	String	Read	The GUID of the user object type. Each user object type must have a unique GUID. The GUID string is in registry format.
Name	String	Read	The user object full name.
Type	String	Read	The user object type. This is defined by the author of the user object type. Typically this type identifies the user object type. Examples: Image, HTML, RTP Media, File, etc.
Description	String	Read/Write	A text description of the user object. You script can change this description if you wish based on your analysis
PreferredFileName	String	Read/Write	Some Unsniff Plugins are very smart. They can figure out the most appropriate name for a user object based on the context in which it was created. For example: The preferred filename of a image transferred via HTML is that of the corresponding GET request. You can change this name if you want based on your analysis.
SenderAddress	String	Read	The network address of the Sender of this User Object. This is a network name if this address has been resolved to a name.
ReceiverAddress	String	Read	The network address of the Receiver of this User Object. This is a network name if this address has been resolved to a name.
StreamID	Long	Read	If this User Object was extracted from a stream. This contains the Stream ID. For user objects not associated with a stream -1 is returned
StreamSeekPos	Long	Read	If this User Object was extracted from a stream. This contains the Stream Seek Position. For user objects not associated with a stream -1 is returned
StreamDirection	String	Read	If this User Object was extracted from a stream. This contains the direction ("in" or

			"out"). For user objects not associated with a stream a null string is returned
Length	Long	Read	The size in bytes of this user object.
HasError	Boolean	Read	Does this user object have an error. Typical errors are when user objects are not completed. You may want to check this property before proceeding to do too much with a given user object.
State	String	Read	The state of the user object.
CreateTimestamp	String	Read	The time this user object was created. The time is returned in a string. The format of the time is determined by the current Windows Locale settings

Methods

Name	Parameters	Description
SaveToFile	FileName (*String*)	Save the user object to a file. *FileName*: Can be a pathname or a relative filename

4 Integrated Scripts

Integrated scripts allow you to add custom functionality to the Unsniff Network Analyzer application. You can attach custom scripts to menu items. Your scripts will be triggered when the user selects the corresponding menu item.

In addition to all the objects you have seen in *Section 3. Scripting Object Model* – you get access to the following objects.

- The currently active capture document.
- The current selection context for packets, PDUs, streams, user objects
- A powerful scripting console that allows you to output formatted text

4.1 Script integration points

You can attach custom scripts to the following menus in Unsniff.

Capture Menu	You can add a menu item under a new top-level menu – or merge with an existing top-level menu such as (Tools, Plugins, Edit). Use this menu if your scripts work on the entire capture file independent of the user selection.
Packet sheet context menu	You can add a menu item to the packet sheet context menu. This context menu is shown when the user right clicks on the packets sheet. Use this is your script needs to work on selected packets.
PDU sheet context menu	Add a menu item to the PDU sheet context menu. Use this method if your script needs to work with selected PDUs.
Streams sheet context menu	Add a menu item to the Streams sheet context menu. This works on entire streams. Use this method if your script needs to work with selected streams.
User Objects sheet Context menu	Add a menu item to the User Objects sheet context menu. Use this method if you want to work with selected user objects.

Script Integration Points

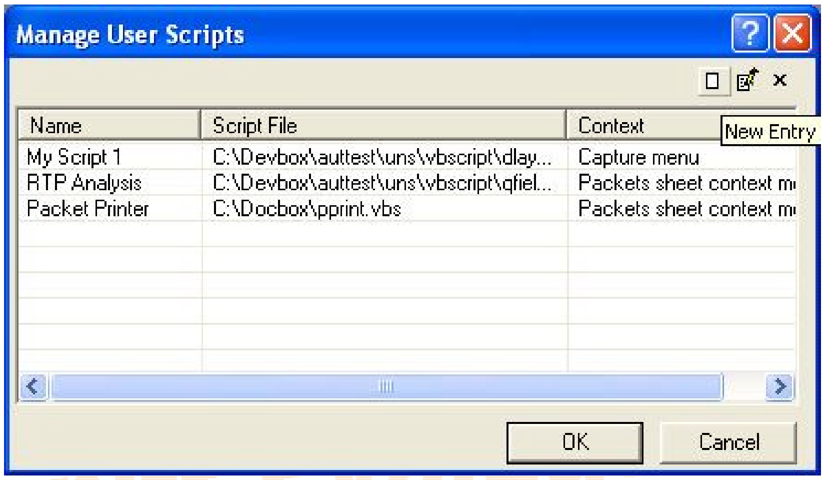
4.1.1 How to integrate scripts into Unsniff

You can integrate your scripts into Unsniff via *Tools -> User Scripts.*

Step-by-step example
Assume that you have written a custom RTP Analysis script. This script analyzes an entire RTP session of a selected RTP packet that is part of the session. In this case you may wish to activate this script from the packets sheet context menu. Here is a step-by-step.

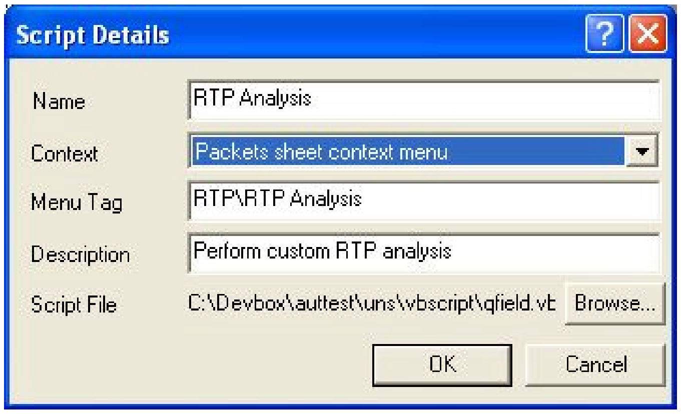
1. Open the User Scripts Manager via *Tools → User Scripts* This opens the "Manage User Scripts" dialog.

2. Click on the "New" button on the top-right corner of the dialog. This button opens the "Script Details Dialog" which allows you to create a new menu item and attach your script to it.



Name	Script File	Context
My Script 1	C:\Devbox\autest\uns\vbscript\dlay...	Capture menu
RTP Analysis	C:\Devbox\autest\uns\vbscript\qfiel...	Packets sheet context m
Packet Printer	C:\Docbox\pprint.vbs	Packets sheet context m

3. The "Script Details" Dialog is shown below. Use this dialog to enter the details shown in the table.

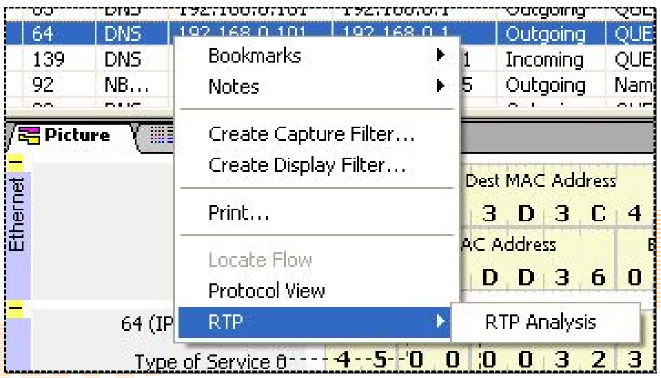


Name: RTP Analysis
Context: Packets sheet context menu
Menu Tag: RTP\RTP Analysis
Description: Perform custom RTP analysis
Script File: C:\Devbox\auttest\uns\vbscript\qfield.vb

Name	A short name for the script functionality
Context	Select where you want to attach your scripts. Five menu options are provided, you need to select one from the drop down list.
Menu Tag	A menu tag identifies how your script will be merged with the existing menu. You can use a “\” (backslash) character to create nested menus. In the above example **RTP\RTP Analysis** is a nested menu. You are encouraged to use nested menus to group related scripts together.
Description	Optional description
Script file	Click the browse button to select your script file. A script file must follow these rules. • VBScript files must have an extension ***.vbs** • Ruby files must have an extension ***.rb** • Jscript files must have an extension ***.js**

Script Details

4. Click OK – then restart Unsniff for your changes to take effect. The figure below shows a custom RTP Analysis script attached to the Packets sheet context menu.



4.2 The CurrentDocument object

Your script will automatically have access to an object called "`CurrentDocument`". This object provides you with access to the currently active capture file as well as the current selection context. Here is a list of properties and method of this object.

Properties

Name	Type	Access	Description
DatabaseName	String	Read	Name of the currently open capture file
Console	Object	Read	Creates a new scripting console object. This can be used to output results of your script. See the next section for a list of properties and methods for the Console object.
PacketCount	Long	Read	Number of packets in the currently open capture file
PacketIndex	Collection	Read	A collection of Packet objects. This represents all the packets in the capture file.
SelectedPacket	Object	Read	The selected packet if a single packet (or) The first selected packet if multiple packets are selected
SelectedPackets	Collection	Read	All selected packets (a collection of Packet objects)
PDUCount	Long	Read	Number of PDUs in the currently open capture file
PDUIndex	Collection	Read	A collection of PDU objects. This represents all the PDUs in the currently active capture file.
SelectedPDU	Object	Read	The selected PDU if single selection (or) The first selected PDU if multiple selection
SelectedPDUs	Collection	Read	All selected PDUs (a collection of PDU objects)
StreamCount	Long	Read	Number of streams in the currently open capture file
StreamIndex	Collection	Read	A collection of Stream objects. This represents all streams in the currently active capture file.
SelectedStream	Object	Read	The selected stream
SelectedStreams	Collection	Read	All selected streams (a collection of Stream objects)
UserObjectsCount	Long	Read	Number of user objects in the currently open capture file
UserObjectsIndex	Collection	Read	A collection of UserObject objects. This represents all the user objects in the currently active capture file.
SelectedUserObject	Object	Read	The selected user object (or) The first selected user object if multiple selection
SelectedUserObjects	Collection	Read	All selected user objects (a collection of UserObject objects)

Methods

This object does not define any methods

4.2.1 The Script Console

Unsniff provides a powerful console via the `CurrentDocument.Console` object. The script console provides rich formatting features that can be used to create great reports. The properties and methods of the Script console are shown below.

Properties

Name	Type	Access	Description
TextColor	String	Read/Write	The current text color. The format of the text color is #RRGGBB. The RGB components are specified in hex. For example: `Con.TextColor = “#FF0000”` Will set the current text color to full red.

Bold	Boolean	Read/Write	The current bold text style. This is a boolean value.
			In VBScript:
			`Con.Bold = True`
			In Ruby:
			`Con.Bold = true`

Hilite	Boolean	Read/Write	The current hilite style. Hilited text appears in a yellow hilite background.
Italics	Boolean	Read/Write	The current italic style.

Methods

Name	Parameters	Description
Write	String	Write the string to the console using the current styles
WriteLine	String	Write the string to the console using the current styles. This method automatically appends the required CR+LF characters. New text will start on a separate line.
SetDefaultFormat	None	Reset all styles. Set TextColor to black.
SetTitle	String	Set the title of the script console window
Clear	None	Clear the contents of the script console window

4.2.2 Example

A simple example will illustrate the use of the CurrentDocument and the Script Console.

Task : Print a description of each selected packet (Packet.Description)

1. Write the following VBScript script and save it to a file (eg: myprint.vbs)

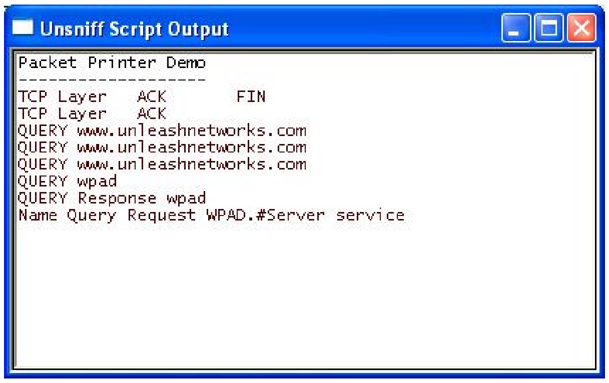
VBScript

```
' ---------------------------------
' Get access to the script console
' ---------------------------------
Dim Con
Set Con = CurrentDocument.Console
Con.TextColor = "#55EE33"

Con.WriteLine "Packet Printer Demo"
Con.WriteLine "--------------------"

Set SelPacketList = CurrentDocument.SelectedPackets

For Each Packet In SelPacketList
      Con.WriteLine     Packet.Description
Next
```

2. Attach the script to the packet sheet context menu. See Section 4.1.1 to find out how you can integrate your script into Unsniff

3. Open a capture file in Unsniff or capture some packets from the network. Then select a few packets from the packets sheet. Right click and select the menu item corresponding to your script.

4. Now the Script Console window will show the desired analysis output.



```
Packet Printer Demo
--------------------
TCP Layer   ACK      FIN
TCP Layer   ACK
QUERY www.unleashnetworks.com
QUERY www.unleashnetworks.com
QUERY www.unleashnetworks.com
QUERY wpad
QUERY Response wpad
Name Query Request WPAD.#Server service
```

<<END>>