scannex

ip.buffer Script Builder

Date	*Author*	*Release*
2009-04-09	MP	Initial release for v1.00

Scannex Electronics Ltd, UK
t: +44(0)8707 48 65 65
f: +44(0)8707 48 67 67

http://www.scannex.co.uk
info@scannex.co.uk

Scannex LLC, USA
t: 1-866-4BUFFER
(1-866-428-3337)

http://www.scannex.com
info@scannex.com

Table of Contents

1. Introduction

The ip.buffer includes the extremely powerful Lua scripting language. Among other things, Lua can be used to analyze, detect, filter and modify incoming call data. However, writing a suitable script from scratch may be a daunting task – especially for those used to using other buffer devices.

The ip.buffer Script Builder is a stand-alone Javascript/HTML document that creates template Lua script for the ip.buffer based on a set of simple inputs.

The Script Builder also includes options to convert DataLink or NetLink style equation expressions into Lua code.

2. Using the Script Builder

2.1. Starting

The Script Builder is deployed as a single .html file. Just double click the "`ip.buffer_ScriptBuilder_x.xx.html`" file. The Script Builder should be shown in your default web-browser.

2.2. Pasting Sample Records

The first area "Sample Records" allows you to paste in some sample call records – to make the job easier. Ideally you should copy a few lines of text from data that you have previously collected.

- Be warned that copying and pasting from the ip.buffer's "LiveView" web page may include marker characters, e.g. "<CR><LF>". Before copying from the ip.buffer web-page you should select "ASCII only" to hide these markers.

The records that you paste will appear under the "Field Display".

2.3. Setting Global Options

The "Global" section allows you to define:

1. A "Name" for the script. Enter a meaningful name, e.g. "SX2000 detect 911 calls". This text is inserted as a comment in the generated script.
2. Select which channel to analyze. In the ip.2 and ip.4 devices you can have separate script functions for each of the collection channels, so you need to specify which channel you want analyzed.

2.4. Creating Field Definitions

The section “Field Definitions” allows the entry of up to 6 separate fields (names “A” to “F”). Each field requires a start column (from 1 upwards) and a length. The “Field Display” section shows column markers to help you work out the positions.

As you enter each field, the “Field Display” will show underneath the positioning of each field. For example, if Field A was defined as starting at column 1, for a length of 5, then “AAAAA” will be shown on the left of the field display.

2.5. Lua Pattern Field Definition

Lua includes some extremely powerful pattern-based string search functions. The Script Builder allows you to enter a single field, called “X”, that uses this power.

1. “Start” - if you enter a value, this tells Lua which column to start looking from. If left blank, then Lua will examine the whole incoming record.
2. “Prefix” - use this to specify an optional prefix string or Lua pattern.
3. “Pattern” - enter a Lua pattern that you are searching for. e.g. “%d+” will search for and extract one or more consecutive digits.
4. “Suffix” - use this to specify an optional suffix string or Lua pattern.

For example, if you need to extract a sequence of digits that are prefixed by an uppercase letter: Prefix=%u, Pattern=%d+

See the Lua manual “Patterns” -
http://www.lua.org/manual/5.1/manual.html#5.4.1

2.6. Equation

The equation is the most important part. It defines when to execute the particular actions that you will choose next. There are three “flavours” or dialects of equation – Lua (native), DataLink, and NetLink.

- All equations are CASE sensitive!

2.6.1. Lua Equations

Enter a Lua compatible equation. The equation may include calls to Lua functions, as well as simple comparisons.

e.g. string.find(txt, '911') and A=='N'

Will match when the digits “911” are found anywhere in the incoming record, and when the field A is the value “N”.

The Script Builder does not check the validity of the equation. However, when you upload to the ip.buffer the ip.buffer will check the syntax and validity (and display an error if anything is wrong).

2.6.2. DataLink Equations

Most DataLink expressions are handled within the Script Builder (including some that may not be handled by the original buffer!). Some examples are shown below:

`A="911"`	Matches when A is the string “911”
`A="9**"`	Matches when A is 9-something-something
`A="911" AND B<>"E"`	Matches when A is “911” and B is not “E”
`@="911"`	Matches when “911” is found anywhere in the record
`@="1**4"`	Matches when 1-something-something-4 is found anywhere in the record
`@<>"1234"`	Matches when “1234” is NOT found in the record
`(A="N" AND B<>"12") OR (A>="E")`	Matches when A is “N” and B is not “12”, or if A is greater than or equal to “E”
`A=B`	Matches when field A equals field B

- As you fill in the equation you will be alerted to basic expression errors. In addition, you can see the generated Lua code being created as you type. This should help in the transition from legacy devices over to the ip.buffer.

2.6.3. NetLink Expressions

Most NetLink expressions are supported in the ip.buffer. Some examples are shown below:

`A=911`	Matches when the field A is “911”
`A=9--&B=N`	Matches when the field A is 9-something-something and B is “N”
`A$911`	Matches when the field A contains the string “911”
`A$1--4`	Matches when the field A contains 1-something-something-4
`A>9|B<5`	Matches when A is greater than “9” and B is less than “5”

- As you fill in the equation you will be alerted to basic expression errors. In addition, you can see the generated Lua code being created as you type. This should help in the transition from legacy devices over to the ip.buffer.

2.7. *On Match*

The "On Match" section defines what will happen when the equation expression matches.

You can do any combination of "Store", "Alert", and "Push".

2.7.1. Store

These functions store data into the ip.buffer memory.

- "Store Text" - entering a value here will store the value in the memory of the channel directly. For example, you may want a tag inserted "`***DETECTED***`". Whenever the equation matches the string will appear in the collected data.
- "Store Record" - choose to store the record that matched. For example, you may want to *only* store records that match the equation. You can select "Store Record = This Channel" and "Always / Store Record = nothing". You may also choose to store the matched records in another storage area (ip.2 and ip.4)

2.7.2. Alert

The alert function will generate an email or HTTP post alert.

- "Alert Tag" - all alerts in the ip.buffer have a text tag that should be short and unique. For example "911call". There should be no spaces in the alert tag.
- "Alert Message" - you can optionally send some informative text along with the alert. For email alerts this text appears in the email body; for HTTP post alerts this is included in the variable set sent to the web-server[1].
- "Alert SNMP Trap" - you can optionally send an SNMP trap out with the alert. Enter a number, e.g. 1001, to specify the specific trap number to be used.

- The ip.buffer Lua implementation includes the ability to have auto-resetting counting alerts; alerts that stay "on" for a set time; and various other combinations. These advanced facilities are outside the scope of the Script Builder, but can be coded directly in Lua.

2.7.3. Push

You can choose to trigger a push delivery of the data for either this channel, or for any other specific channel number.

[1] The Script Builder will create a script that includes a static, fixed, message. However, Lua allows you to include information about the record itself. For example, you could change the generated Lua script from "`alert.alarm("AlertTag","Testing")`" to "`alert.alarm("AlertTag","Testing:"..A..B)`" to append field A and B to the message.

2.8. Always section

The "Always" section specifies what should happen to the record every time a record is received (that is, even if the equation does not match).

- "Store Record" - you can choose to store the record in this channel or any other specific channel, or just do nothing (i.e. discard the record).

- In actual fact, the "Always" action is performed *before* the equation test. This allows the "Always" action to store the data, and for the "On Match" to trigger a push delivery with all the data included.

2.9. Script Result

The "Script Result" shows the generated Lua script. This script can be copied and pasted into the ip.buffer's "Script" area (Setup / Advanced / Script / Edit).

- The ip.buffer's source protocol should be set to "`ASCII Lines`". It is possible to write Lua scripts that handle multi-line protocols (like Nortel Norstar), but this is outside the scope of the ScriptBuilder tool.

3. Suggestions and Hints

The Script Builder will quickly show you how to create correct Lua expressions and code. Many applications will be completely solved using just the Script Builder. However, more complex requirements may involve writing Lua (and using the Script Builder to help).

3.1. Case Sensitivity

The Lua language is case sensitive. Thus, "Function" is different from "function". Pay particular attention to this! You may find it helpful to write any variables you assign in CAPITAL letters.

3.2. Variable Scope

You will see that the Script Builder defines the fields with the prefix "local". The "local" prefix ensures that variables are specific to the "ProcessRecord" function under which they are created.

Missing out the "local" keyword can create havoc when there are functions for multiple channels!

3.3. Multiple Channels

The last line of the Script Builder generated Lua code is in the form "x.chnl[n].src.onrecord=...".

You can "glue" as many channels to either the same function, or to different functions.

e.g.

```
x.chnl[1].src.onrecord=ProcessRecord

x.chnl[2].src.onrecord=ProcessRecord
```

Will perform the same record handling for both Channel 1 and Channel 2.

3.4. Multiple Expressions

You can include as many "if … then … end" expressions as you need.

The Script Builder only generates one, but you can create a compound Lua script based on any number of equation expressions.

3.5. Going further with Lua

Lua has many excellent string extraction, detection, and pattern matching functions that make it easy to perform very complex record analysis.

Once the Script Builder has given you the "feel" for Lua coding, check out the documentation links on the Lua website: www.lua.org

4. Suggestions for Testing

Before deploying the script in the field it is good to test the script. Setting up the ip.buffer in the following way makes it relatively easy to test:

- Download and install SETelnet on a PC (available from www.scannex.com)
- Set the ip.buffer up:
    - Source
        - Collect from TCP
            - Set “Connect” to “`Device to ipbuffer (passive/server)`”
            - Set the port to a known value, e.g. “`2001`”
            - Clear all the “Match & Send”, and “Heartbeat” fields
        - Protocol “`ASCII Lines`”, no time stamp
    - Destination
        - Deliver to “`TCP server (passive)`”
            - Set the port to a known value, e.g. “`5001`”
            - Clear the password
            - Set “On Complete” to “`Stay connected (real-time)`”
- Now run two copies of SETelnet:
    - First one connect to port 2001 (i.e. the source port)
    - Second one connect to port 5001 (i.e. the destination port)

With this setup you can now send data into the source port and see what is stored and delivered in real time on the destination port.

SETelnet allows you to send a file straight from disk.

- If you reload the Lua script and reboot Lua then the source will disconnect automatically. You will need to click “Close” and then “Open” on the SETelnet that was connected to the source.