53-1002445-01
January, 2012

ServerIron ADX

OpenScript Programmer’s Guide

Supporting Brocade ServerIron ADX version 12.4.00

BROCADE

Brocade Communications Systems, Incorporated

Corporate and Latin American Headquarters
Brocade Communications Systems, Inc.
130 Holger Way
San Jose, CA 95134
Tel: 1-408-333-8000
Fax: 1-408-333-8101
E-mail: info@brocade.com

Asia-Pacific Headquarters
Brocade Communications Systems China HK, Ltd.
No. 1 Guanghua Road
Chao Yang District
Units 2718 and 2818
Beijing 100020, China
Tel: +8610 6588 8888
Fax: +8610 6588 9999
E-mail: china-info@brocade.com

European Headquarters
Brocade Communications Switzerland Sàrl
Centre Swissair
Tour B - 4ème étage
29, Route de l'Aéroport
Case Postale 105
CH-1215 Genève 15
Switzerland
Tel: +41 22 799 5640
Fax: +41 22 799 5641
E-mail: emea-info@brocade.com

Asia-Pacific Headquarters
Brocade Communications Systems Co., Ltd. (Shenzhen WFOE)
Citic Plaza
No. 233 Tian He Road North
Unit 1308 – 13th Floor
Guangzhou, China
Tel: +8620 3891 2000
Fax: +8620 3891 2111
E-mail: china-info@brocade.com

Document History

Title	Publication number	Summary of changes	Date
ServerIron ADX OpenScript Guide	*53-1002445-01*	New document	January, 2012

Contents

About This Document

Supported hardware and software

Although many different software and hardware configurations are tested and supported by Brocade Communications Systems, Inc. for 12.4.00 documenting all possible configurations and scenarios is beyond the scope of this document.

The following hardware platforms are supported by this release of this guide:

- ServerIron ADX 1000
- ServerIron ADX 4000
- ServerIron ADX 10K

Document conventions

This section describes text formatting conventions and important notice formats used in this document.

Text formatting

The narrative-text formatting conventions that are used are as follows:

bold text	Identifies command names
	Identifies the names of user-manipulated GUI elements
	Identifies keywords
	Identifies text to enter at the GUI or CLI
italic text	Provides emphasis
	Identifies variables
	Identifies document titles
`code` text	Identifies CLI output

For readability, command names in the narrative portions of this guide are presented in bold: for example, **show version**.

Notes, cautions, and danger notices

The following notices and statements are used in this manual. They are listed below in order of increasing severity of potential hazards.

NOTE
A note provides a tip, guidance or advice, emphasizes important information, or provides a reference to related information.

CAUTION
A Caution statement alerts you to situations that can be potentially hazardous to you or cause damage to hardware, firmware, software, or data.

DANGER
A Danger statement indicates conditions or situations that can be potentially lethal or extremely hazardous to you. Safety labels are also attached directly to products to warn of these conditions or situations.

Notice to the reader

This document may contain references to the trademarks of the following corporations. These trademarks are the properties of their respective companies and corporations.

These references are made for informational purposes only.

Corporation	Referenced Trademarks and Products
Microsoft Corporation	Windows NT, Windows 2000
The Open Group	Linux

Related publications

The following Brocade documents supplement the information in this guide:

- *Release Notes for ServerIron ADX 12.4.00 release*
- *ServerIron ADX Administration Guide*
- *ServerIron ADX Advanced Server Load Balancing Guide*
- *ServerIron ADX Firewall Load Balancing Guide*
- *ServerIron ADX Global Server Load Balancing Guide*
- *ServerIron ADX Graphical User Interface*
- *ServerIron ADX Hardware Installation Guide*
- *ServerIron ADX Security Guide*
- *ServerIron ADX Server Load Balancing Guide*
- *ServerIron ADX Switch and Router Guide*
- *ServerIron ADX XML API Programmers Guide*

- *ServerIron ADX NAT64 Configuration Guide*
- *ServerIron ADX OpenScript API Guide*
- *IronWare MIB Reference*

Getting technical help or reporting errors

Brocade is committed to ensuring that your investment in our products remains cost-effective. If you need assistance, or find errors in the manuals, contact Brocade using one of the following options:

Web access

The Knowledge Portal (KP) contains the latest version of this guide and other user guides for the product. You can also report errors on the KP.

Log in to my.Brocade.com, click the **Product Documentation** tab, then click on the link to the Knowledge Portal (KP). Then click on **Cases** > **Create a New Ticket** to report an error. Make sure you specify the document title in the ticket description.

E-mail and telephone access

Go to http://www.brocade.com/services-support/index.page for the latest e-mail and telephone contact information.

Chapter

Overview of OpenScript 1

The Application Delivery environment requires more than simple CLI commands for managing Application traffic. Often, an operator wants to make packet forwarding decisions based on real-time events such as layer-3, layer-4, layer-7 data or server metrics such as current server load statistics. These situations require a more dynamically programmable environment than traditionally offered through built-in CLI commands.

In essence, the operators prefer programming the Application Forwarding Behavior rather than statically provisioning it. The flexibility of programming the data-plane at layer-7 is invaluable in dynamic data center environments as new applications come online.

The CLI-based interface of ServerIron ADX currently consists of a list of policy, rule, and configuration commands that can be used to configure the application delivery switch for such tasks as real server configuration, load-balancing metrics, as well as for traffic redirection and transformation operations.

What is OpenScript

Brocade OpenScript provides data plane scripting functionality to manipulate traffic in real-time.The Brocade OpenScript engine provides a programming framework and protocol level APIs that allow you to create Perl scripts to customize the traffic handling capabilities of a ServerIron ADX. With this capability, you do not have to depend on Brocade to add needed functionality to suit the needs of your network.

OpenScript is available beginning with ServerIron ADX version 12.4.00. No separate license is required and version 12.4.00 is available to all customers with an active service contract.

Why OpenScript uses Perl

Perl was chosen for OpenScript for the following primary reasons:

- Performance
- Extensibility
- Ubiquity

Performance

Because of the need to compile and interpret the program, scripting is an inherently slow process in regards to performance. Consequently, scripting is often implemented as a multipass process. This is preferable to the large overhead associated with the single pass execution model used with command-based languages such as TCL that parse and execute each program statement on the fly. In a multipass Scripting Engine such as Perl, the initial phase involves scanning and parsing the input script to generate an intermediate byte code representation. The final run phase executes the generated byte code by invoking underlying machine operations associated with each node in the parsed byte code tree.

Extensibility

Perl was selected for the OpenScript platform because it is open and modular which allows new functionality to be added easily. In addition to the support provided by Brocade, CPAN (Comprehensive Perl Archive Network), is one of the largest repositories of free code in the world. If you need a particular type of functionality, chances are there are several options on the CPAN, and there are no fees or ongoing costs for using it. While it's possible to download one or more CPAN Perl modules and install them via a user script on a version updates, the users must do so entirely at their own risk. We recommend using Perl Extensions that either come pre-loaded with the ServerIron ADX or approved by Brocade.

Ubiquity

Perl is one of the most heavily used scripting languages for a multitude of applications from text processing to Apache web server traffic processing. It is easy to use, efficient and complete. ServerIron ADX-specific functionality such as server load balancing and content transformation within Perl are exposed to the user via custom extensions that are familiar and easy to use. The extension interface for ServerIron ADX Perl modules has been designed to mirror the functionality available on CPAN. The difference is that ServerIron ADX Perl Extensions are optimized for most common use cases and hide the backend complexity from the script writer. Perl also has an extensive user and developer community to publish and share scripts covering popular deployment scenarios.

Why OpenScript uses Perl instead of TCL

Besides Perl, TCL is another scripting platform that is popular in the Industry and applicable for ServerIron ADX. TCL is a well established scripting platform that finds popular use in the embedded device environment. Factors favoring TCL are familiarity, stability and extensibility. It offers a C API to extend scripting functionality via native C code.

The primary concern with embedding TCL is performance. Like the UNIX shell, TCL is a purely interpretive language. This means that TCL is interpreted each step along the way. Although TCL is run with a small and efficient interpreter, Perl which is first compiled and then run via an intermediate byte code representation is inherently faster at execution.

Table 1 describes the advantages and disadvantages to using Perl and TCL.

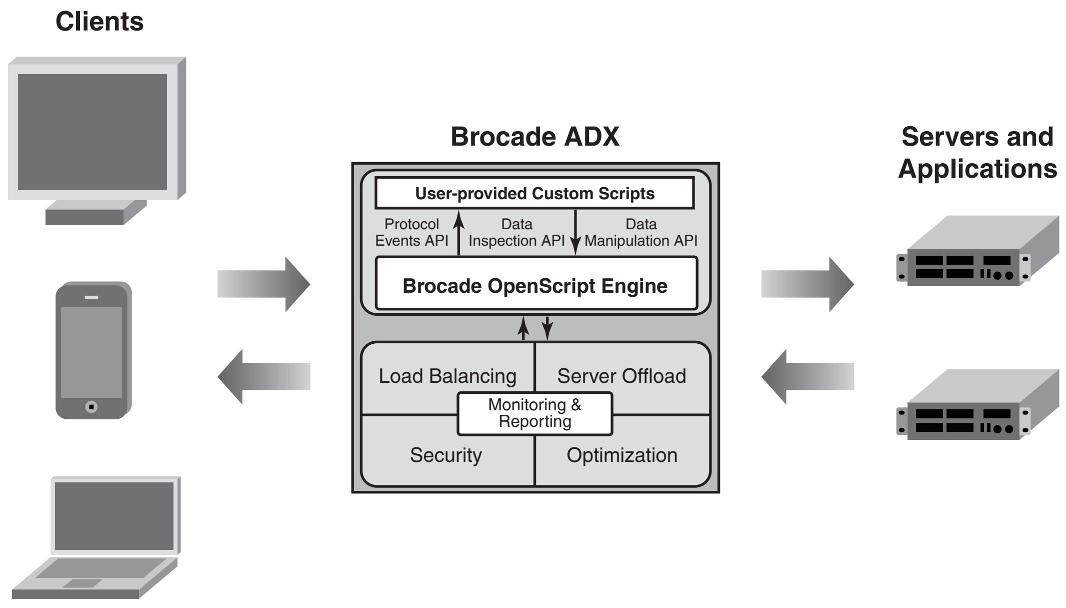
TABLE 1 comparison between Perl and TCL

	Perl	Tcl
Advantages	Rich language features. Rich libraries (CPAN) Fast performance with the compile and run module	Popular with Server Load Balancing vendors.
Disadvantages	Bulkier with a larger dependency baggage.	Performance lags because Tcl is a pure interpretive language.

The OpenScript Engine

The OpenScript engine provides the ability for direct interaction with traffic passing through a ServerIron ADX. User-provided, custom logic written in the Perl programming language can use ServerIron ADX monitoring capabilities to observe network traffic between clients and servers and then react to traffic patterns by altering the traffic flows. Figure 1 illustrates the flow of traffic through an OpenScript engine residing on a ServerIron ADX. As traffic moves between clients and servers, the OpenScript engine can respond to changing traffic patterns and shape traffic flows as required.

FIGURE 1 The OpenScript Engine



As shown in Figure 1, the OpenScript engine can host user-provided custom Perl scripts that use protocol events APIs and Data inspection APIs to determine traffic conditions and then call data manipulation APIs to change traffic flows.

Capabilities provided by the Brocade OpenScript engine

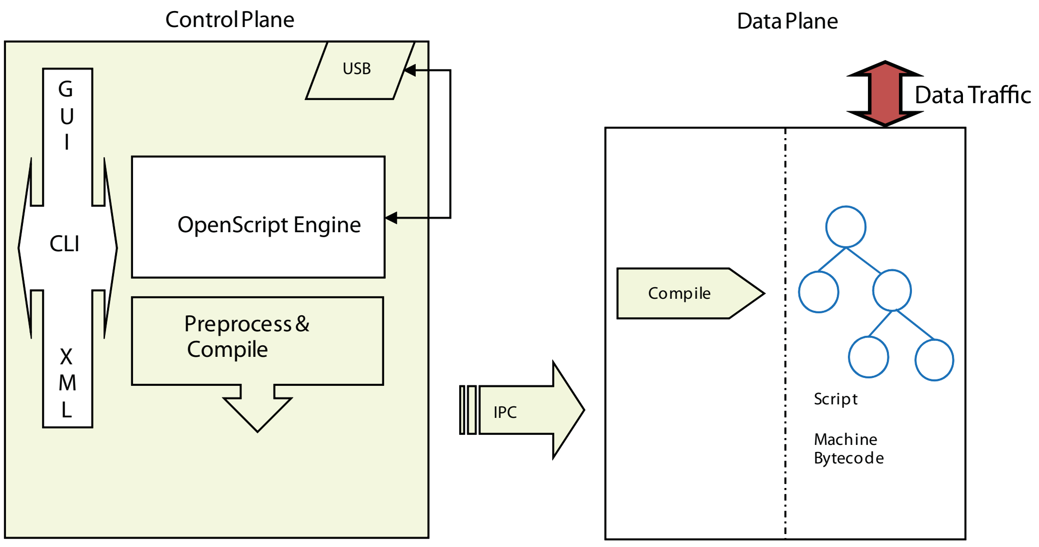
Using the capabilities of the Brocade OpenScript engine, you can enhance the operation of the ServerIron ADX to provide the following:

- Future-proof your infrastructure by transforming ServerIron ADX functionality without changing hardware or software.
- Facilitate rapid delivery of new services to end users
- Fix broken applications
- Monitor traffic with custom statistics and send it to data collectors
- Deliver legacy applications that are not supported on the ServerIron ADX natively
- Temporarily mitigate security attacks through rapid script development

Architecture of the OpenScript engine

Because script parsing is highly CPU-intensive it is performed entirely on the management processor (MP) of the ServerIron ADX. If the compilation succeeds on the MP, the script is downloaded to the application processor (BP) for installation. The BP prepares the script by generating machine byte code and binds it by inlining it in the packet processing path for the virtual server and service. This process is displayed in Figure 2.

FIGURE 2 OpenScript compilation and installation process



Performance estimator

The scripting engine provides the capability to predict the performance impact of a given script within a 25% margin of error. The estimate is provided as a measure of CPS and TPS percent degradation from the published performance baseline.

Using Perl on the ServerIron ADX

The Perl implementation on ServerIron ADX is based on Perl Version 5.12. It preserves the core syntax and semantics of the core language as specified at the perldoc site: http://perldoc.perl.org/5.12.4/perlsyn.html.

Perl variables

Perl variable scoping on ServerIron ADX

- Any lexical variable declared as “my” within a script is re-initialized on every run. Variable scoping is limited to the script it resides in. This means that variables are not visible to other scripts.
- A state lexical variable behaves like a C language static, with persistent value across reruns. It is never re-initialized like “my”.

TABLE 2 Perl lexical variable scoping on a ServerIron ADXs

Lexical Type / object	Scope	Re-initialized per run	Exported to MP	Use	Data Type Allowed	Limit
my	script	Yes	No	Script local / auto	Perl all	Script
state	script	No	No	Script static	Perl all	Script
Conn hash	connection	No	Yes	Correlate client & server flows	Perl all	Script

Chapter

OpenScript Fundamentals 2

Overview

This following sections of this chapter describe the process of creating a simple script using the Perl-based, OpenScript environment.

Structure of a ServerIron ADX Perl script

The structure of a Perl script written for OpenScript differs slightly from the standard free-flowing script program usually associated with Perl. In a regular perl script, methods (subs) can be freely defined and invoked from the main body of the script. To provide for optimal packet processing, scripts written for the OpenScript environment build their behavior around the scripting engine with application and protocol event handlers. These handlers are a published list of events that a user script can catch by enclosing their event handling code in a Perl function or sub named after the event. Figure 3 provides a list of application and protocol events provided for OpenScript. For a complete list of events, see the *Brocade OpenScript API Reference Guide*.

TABLE 3 Application and Protocol Events in ADX scripts

Application/ Protocol	Event Description	Attached Script Method	Use
HTTP	On request header parsing completion	HTTP_REQUEST	Perform request header inspection and insertion/deletion
	Request payload data available. Called when entire payload available. **Only triggered by collect() API.**	HTTP_REQUEST_DATA	Inspect and modify HTTP request payload data.
	On response header parsing completion	HTTP_RESPONSE	Perform response header inspection and insertion/deletion.
	Response payload data available. Called when entire response payload is available. **Only triggered by collect() API.**	HTTP_RESPONSE_DATA	Inspect and modify HTTP response payload.
TCP	On new client SYN request.	TCP_CLIENT_SYN	Authentication, connection rate-limiting based on IP.
	On successful TCP client connection establishment.	TCP_CLIENT_ESTABLISHED	Pre TCP data transfer processing context.
	On client initiating a TCP CLOSE.	TCP_CLIENT_CLOSE	Update counters/state.
	On receiving a TCP client RESET	TCP_CLIENT_RESET	Update counters/state.

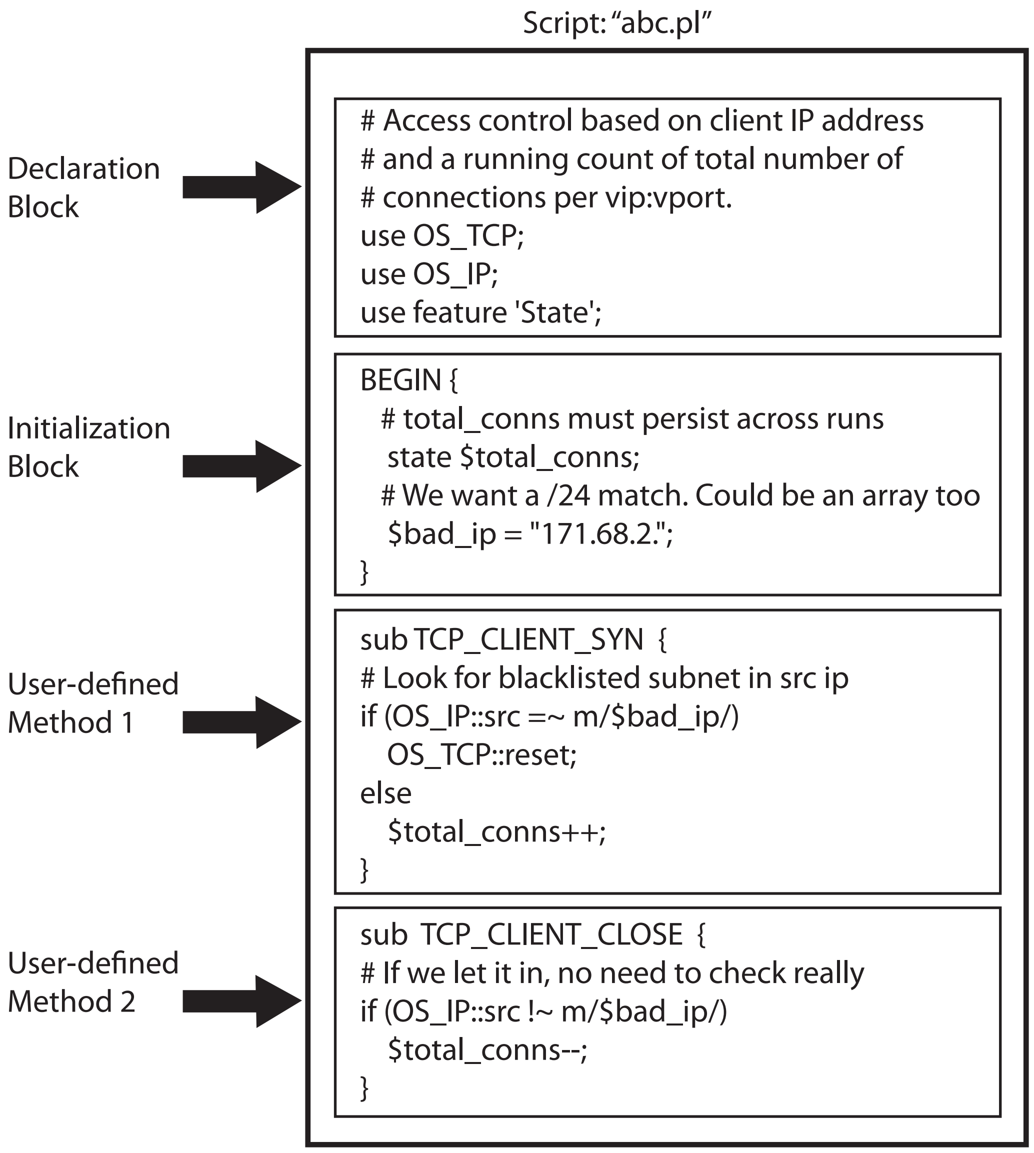
TABLE 3 Application and Protocol Events in ADX scripts

Application/ Protocol	Event Description	Attached Script Method	Use
	On TCP client layer 7 application payload data being available. **Only triggered by collect() API**	TCP_CLIENT_DATA	Inspect and transform client side TCP application data.
	On server initiating a TCP CLOSE.	TCP_SERVER_CLOSE	Update counters/state.
	On receiving TCP RESET from server.	TCP_SERVER_RESET	Handle server connection resets.
	On successful connection establishment with server	TCP_SERVER_ESTABLISHED	Packet processing context before sending client data to server.
	On TCP server layer 7 application payload data being available. **Only triggered by collect() API.**	TCP_SERVER_DATA	Inspect and transform server side TCP application data.
UDP	On UDP client layer 7 application payload data being available.	UDP_CLIENT_DATA	Inspect and transform UDP client side application data.
	On UDP server layer 7 application payload data being available.	UDP_SERVER_DATA	Inspect and transform UDP server side application data.
SLB	On server selection failure.	SERVER_SELECTION_FAILURE	Customize action for recovery.

Basic anatomy of a script

The basic example script (abc.pl) is designed to exercise access control based on a client's IP address and a running count of the total number of connections per virtual server port. As displayed, it consists of the following elements:

- Declaration Block – Declares the packages being used by the script.
- Initialization block – Only evaluated once before the first run of the script
- User-defined method 1 – Method 1 is designed to handle a new TCP client connection request. It is invoked on every TCP SYN received on the vport bind point.
- User-defined method 2– Method 2 is designed to run on receiving a TCP CLOSE request from a client.



```
# Access control based on client IP address
# and a running count of total number of
# connections per vip:vport.
use OS_TCP;
use OS_IP;
use feature 'State';

BEGIN {
   # total_conns must persist across runs
    state $total_conns;
   # We want a /24 match. Could be an array too
    $bad_ip = "171.68.2.";
}

sub TCP_CLIENT_SYN  {
# Look for blacklisted subnet in src ip
if (OS_IP::src =~ m/$bad_ip/)
    OS_TCP::reset;
else
    $total_conns++;
}

sub  TCP_CLIENT_CLOSE  {
# If we let it in, no need to check really
if (OS_IP::src !~ m/$bad_ip/)
    $total_conns--;
}
```

Sample scripts

The following examples provide two different approaches to creating a script for the same purpose. The first example provides a heavily commented example for high readability and the second is a “power-user” version of the script. It is much more compact with less extensive notation. Both scripts provide for load-balancing using a URL match in an HTTP GET request.

High readability script example

The following example provides a version of the script that demonstrates high-readability.

```
# Elegant load balancer with good readability
# Performs server selection based on URI in HTTP
# Request Header

# Tell the script about extensions used in this program
use OS_HTTP_Request;
use OS_SLB;

# Define an event handler to be called when an HTTP
# request is received
sub HTTP_REQUEST
{
    #Get an HTTP request object to manage
    # request headers or content.
    $request = OS_HTTP_REQUEST::get;

    # Pattern we want to match…..
    $url_pattern = "index.html";
    # …. with pattern data in packet
    $url_in_request = $request->url;

    # Perform a Regular Expression Search.
    if ($url_in_request =~ m/$uri_pattern/) {
        # Forward to a real server identified by name
        OS_SLB::forward("RS1");
    } else {
        # Forward to a real-server group identified by numeric ID
        OS_SLB::forward(2);
    }
}
```

Power-user script example

The following example is the “power-user” version of the script. It is much more compact with less extensive notation.

Example 2:

```
# Power User version
# Performs server selection based on URI in
# HTTP Request Header

use OS_HTTP_Request;
use OS_SLB;

sub HTTP_REQUEST
{
    # local variable with default server group-id
    $server = 2;
    $request = OS_HTTP_REQUEST::get;
    if ($request->url =~ m/"index.html"/)
         $server = "RS1";

    # $server can hold integer or string values.
    # Xtension backend does translation.
    OS_SLB::forward($server);
}
```

Chapter

Managing Scripts on a ServerIron ADX 3

Overview

A script can be written with any text editor or using the ServerIron ADX GUI. The ServerIron ADX GUI.process is described in the *ServerIron ADX GUI Configuration Guide*.

Once a script has been written, it must be uploaded to a ServerIron ADX to be compiled and bound to a port. In addition, several operations can be performed on the script and a profile can be defined that sets the environmental variables under which the script will run. This chapter describes each of these operations.

- "Importing and exporting scripts" – This section describes the processes for importing a script onto a ServerIron ADX or exporting a script from a ServerIron ADX using TFTP.
- "Managing scripts" This section describes how to delete or rename a script that you have imported onto a ServerIron ADX. It also describes the process of displaying the contents of a script that resides on a ServerIron ADX.
- "Compiling and binding scripts" – Once a script has been imported to a ServerIron ADX, it must be compiled and bound to a virtual server port. This section describes how to perform these actions as well as how to obtain a performance estimate of a script and update the contents of an existing script.
- "Creating and configuring script profiles" – A series of default values for environmental variables are used in compilation of a script on a ServerIron ADX. If these values are acceptable for your use, you do not need to create and define a script profile. This section describes the default values and how to create and define a script profile that alters these values for your application.
- "Displaying script information" – Several show commands are provided to display information about scripts residing on a ServerIron ADX.

Importing and exporting scripts

For a script to be run on a ServerIron ADX, it must be uploaded to the following file location:

usb0\sys\dpscript

A script can be uploaded using any of the following methods.

- through the ServerIron ADX GUI. This process is described in the *ServerIron ADX GUI Configuration Guide*.
- Using TFTP as shown in "Importing and exporting scripts through TFTP"

Script name size limit

A script has a name and extension. The size of the name can't exceed 8 characters. the size of the extension can't exceed 3 characters.

Importing and exporting scripts through TFTP

You can use the **copy tftp** command to upload a script to a ServerIron ADX from a TFTP server as shown.

```
ServerIronADX# copy tftp usb0 1.1.1.1 sample.pl sys\dpscript\sample.pl
```

Syntax: **copy tftp usb0** <*IP_address*> <*script_filename_*on*_tftp-server*> **sys\dpscript\** <*filename_on_usb0*>

The <*IP_address*> variable is the address of the TFTP server where the script resides.

The <*script_filename_on_tftp-server*> variable is the filename of the script file.

The <*filename_on_usb0*> variable is the script name to be created on the ServerIron ADX.

Scripts can also be exported from a ServerIron ADX to a TFTP server using the **copy usb0 tftp** command as shown in the following.

```
ServerIronADX# copy usb0 tftp 1.1.1.1 sys\dpscript\sample.pl sample.pl
```

Syntax: **copy usb0 tftp** <*IP_address*> **sys\dpscript\**<*filename_on_usb0*> <*script_filename_on_tftp-server*>

The <*IP_address*> variable is the address of the TFTP server that you want to copy the script to.

The <*filename_on_usb0*> variable is the script name on the ServerIron ADX that you want to copy to the TFTP server.

The <*script_filename_on_tftp-server*> variable is the script name to be created on the TFTP server.

Managing scripts

In addition, the you can manage scripts on the ServerIron ADX as described in the following.

- "Deleting a script"
- "Renaming a script"
- "Display script in the script directory"

Deleting a script

You can use the **delete** command to delete a script from a ServerIron ADX as shown.

```
ServerIronADX# delete usb0\sys\dpscript\sample.pl
```

Syntax: **delete usb0\sys\dpscript\** <*script-name*>

The <*script_name*> variable is the name of the script file that you want to delete.

NOTE
You cannot delete a script if it is bound to a VIP.

Renaming a script

You can use the **rename** command to rename a script on a ServerIron ADX as shown.

```
ServerIronADX# rename usb0\sys\dpscript\sample.pl usb0\sys\dpscript\sample1.pl
```

Syntax: rename usb0\sys\dpscript\ <*initial-script-name*> **usb0\sys\dpscript\** <*new-script-name*>

The <*initial-script-name*> variable is the name of the script file that you want to rename.

The <*new-script-name*> variable is the new name that you want to create for the script file.

NOTE
A script that is currently bound to a VIP port cannot be renamed.

Display script in the script directory

You can use the **dir** command to display all of the scripts in the script directory as shown.

```
ServerIronADX# dir usb0\sys\dpscript
Name                                        Size      Type     Date
.                                              0      [Di]     10/06/2011 01:30:02
..                                             0      [Di]     10/06/2011 01:30:02
READ_IP.PL                                  2113      [A ]     10/29/2011 01:22:02
HTTP_1.PL                                   4412      [A ]     10/29/2011 01:32:00
      Size : 4102352896 bytes
Bytes Used :  249810944 bytes
Bytes Free : 3852541952 bytes
ServerIronADX
```

Syntax: dir usb0\sys\dpscript

Compiling and binding scripts

Once a script has been copied to ServerIron ADX you must compile it before it can be run. After the script is compiled, it is ready to run on the ServerIron ADX but it must be bound to a VIP that you it to operate on. Unless a script is explicitly bound to a VIP it will not effect traffic on the ServerIron ADX.

This script compile can be done by itself, within the bind operation, or within the process of updating an existing script. The following is the recommended sequence for binding and maintaining a script:

1. "Compiling a script and obtaining output from the performance estimator" – We recommend that you compile a new script using the **script compile** command before binding it to a virtual server port. This allows you to make sure that the script compiles successfully and gives you an estimate of script performance using the Performance Estimator. Once the script is performing up to your expectation, you can run bind it to the virtual server port.
2. "Binding a script to a virtual server port" – The script binding operation compiles the script unconditionally and then enables it for packet processing on a specified server port or service. If compilation errors occur, they are printed to the console or syslog and the bind operation is aborted.
3. "Updating an existing script"– Once a script has been running, you can make changes to it and replace it all instances of it in a running configuration. This process compiles the script unconditionally and if the compile is successful, unbinds the old version from previously bound virtual server ports and then binds the new script to the same ports.

Compiling a script and obtaining output from the performance estimator

You can compile a script as a single independent operation through the **script compile** command. Running this command insures that the compile will be successful and displays the results of the performance estimator.The following example compiles the “slb.pl” script and runs the performance estimator on it.

```
ServerIronADX(config)# script compile slb.pl
  Performance for this script:
       -  Approximately <123> connections/second at 10% CPU utilization
       -  Approximately <1234> connections/second at 50% CPU utilization
       -  Approximately <12345> connections/second at 100% CPU utilization
```

NOTE
Script compile can only be performed in general config mode.

Syntax: **script compile** <*script-name*>

The <*script_name*> variable is the name of the script that you want to compile.

This compile operation performs language checks on the supplied script. If errors are encountered, they are displayed on the console. A successful compile operation qualifies a script as ready-to-run and it can the be bound to a virtual server port.

Performance estimator

As you can see in the example of the **script compile slb.pl** command above, a script performance estimate is provided for running the script on a ServerIron ADX. This is done to provide guidance concerning operation of the script, This performance estimate is provided in connections-per-second. It is printed out as part of the output of the script compile command for three CPU usage scenarios: 10%, 50% and 100% CPU utilization,

NOTE
The result is estimated given a simple traffic pattern, the real number may vary from this estimate.

Binding a script to a virtual server port

The script binding operation compiles the script unconditionally and then enables it for packet processing on a specified server port or service.

The update is performed as shown.

```
ServerIronADX(config)# server virtual-name-or-ip vs1 100.1.5.101
ServerIronADX(config-vs-vs1)# port http script sample.pl script-profile sp1
```

Syntax: **[no] port** <*port-num-or-service*> **script** <*script-name*> **[script profile** <*script-profile-name*> **]**

The <*port-num-or-service*> variable species the virtual server port or service that you want to bind the script to.

The <*script_name*> variable is the name of the script that you compile and bind.

The **script profile** parameter directs the ServerIron ADX to apply the previously configured script profile specified by the <*script-profile-name*> variable to the script being bound. If you do not specify a script profile the default script profile values will be used. See “Creating and configuring script profiles” on page 17.

Updating an existing script

You can update all running instances of a script with the contents a newly updated script of the same name. The **script update** command allows you to refresh all instances of a script bound to active servers, after the script has been edited. Running this command compiles the script, unbinds the current instance and then binds the new version of the script. If the compile fails, the update is not completed.

The update is performed as shown.

```
ServerIronADX(config)# script update sample.pl
```

Syntax: **script update** <*script-name*>

The <*script_name*> variable is the name of the script that you want to update all running instances of.

Creating and configuring script profiles

A script profile sets the environmental variables for any script it is applied to. These variables involve use of memory by the script and behavior during script operation. This is an optional configuration, Where a script profile is not configured, the default values will take effect.

The environmental variables that can be set with a script profile and their default values are described in the following:

- **script memory limit**: System memory is dynamically allocated from a ServerIron ADX for use by a script. Because the system memory used by a script is shared with all other services provided by the ServerIron ADX, excessive memory use by a script can impede system operation. You can set a limit on the amount of system memory used by a script. Setting a limit causes operation of a script to be halted if the limit set is exceeded. The default value is 1M bytes (1,048,576 bytes).
- **script memory high watermark:** You can configure the ServerIron ADX to generate a syslog message if the memory usage of a script exceeds a percentage that you configure. This serves to inform you that a script is near the “script memory limit” before the script is halted. The default value is 90%
- **script timeout:** Because scripts written for OpenScript are designed to be event-driven, they make repeat runs until an event or threshold that is being monitored for occurs and a action is implemented. If the first run through of a scrip takes too much time, it is an indication that the script is not running properly. The script timeout sets a watchdog time in milliseconds that will halt the script if is exceeded during the first run. The default value is 200 (milliseconds).
- **script data collection limit:** A script can be written for OpenScript that accumulates data for specified events. This data is stored in system memory that is shared with all other services provided by the ServerIron ADX. Because this memory is shared with all other services provided by the ServerIron ADX, excessive memory used in a data collection script can impede operation. If this limit is reached the script is restarted and memory used for data connection is cleared. The default value is 100000 (100 Kbytes).

- **script restart limit:** If a script is having problems during operation or it exceeds limitations set in the profile (either default or configured) it will restart. This option can be configured to limit the number of times that the script will be restarted. Once this limit is reached, the script will be halted and not restarted. After exceeding this limit, the script can only be restarted by either unbinding and rebinding the script or by restarting the ServerIron ADX. By default there is no limit to the number of script restarts.
- **script debugging:** With the option turned on, the script will print debugging information to the console. Debugging information can only be printed to the console. It is turned **off** by default.
- **script print output:** By default the print function in a script is directed to print to the console This option allows you to direct the script to print to either the console, the syslog or to disable the script from printing.

Creating a script profile

The script profile must be named before you can define the script parameters and the name can then be used to re-enter the profile configuration to change the settings for an existing profile You can create a script profile or enter the configuration mode for an existing script profile as shown.

```
ServerIronADX(config)# script-profile sp1
ServerIronADX(config-script-profile-sp1)#
```

Syntax: **[no] script profile** <*profile-name*>

The <*profile-name*> variable is the name of the profile that you want to create or update.

Using the **no** parameter before the command deletes the named profile.

NOTE
When an script profile is changed, the updated profile parameters are automatically applied to the virtual ports that it is bound to.

Setting the script memory limit

This parameter sets the memory limit for any script that is bound to it. When the memory limit is decreased, the user is prompted to confirm the reduction. The script memory limit is set as shown.

```
ServerIronADX(config)# script-profile sp1
ServerIronADX(config-script-profile-sp1)# memory-limit 200000
Are you sure you want to decrease the memory limit from 1048576 to 200000? (yes or
no): yes
```

Syntax: **[no] memory-limit** <*size*>

The <*size*> variable is new limit that you want to set. The default value is 1M bytes (1,048,576 bytes). While you can set this variable to any value that fits within the size of the system memory, all memory limits set for scripts are subject to their impact on system performance

Using the **no** parameter before the command returns the memory limit to the default value.

Setting the script memory high-water mark

This parameter sets the script memory high-water mark percentage. When the memory high-water mark percentage is reached, a syslog message is generated and the script resets the new connections. The script memory high-water mark percentage is set as shown.

```
ServerIronADX(config)# script-profile sp1
ServerIronADX(config-script-profile-sp1)# memory-high-watermark 80
Are you sure you want to dercres the memory limit from 1048576 to 200000? (yes or
no): yes
```

Syntax: **[no] memory-high-watermark** <*percentage*>

The <*percentage*> variable is the new high-water mark percentage that you want to set. The default value is 90 (%).

Using the **no** parameter before the command returns the memory high-water mark to the default value.

Setting the script timeout

Use this command to set the time in milliseconds that you want to allow a script for its first run before restarting it. The following example sets the watchdog time to 1000 milliseconds for the "sp1" script.

```
ServerIronADX(config)# script-profile sp1
ServerIronADX(config-script-profile-sp1)# timeout-ms 1000
```

Syntax: **[no] timeout-ms** <*milliseconds*>

The <*milliseconds*> variable is new watchdog time that you want to set in milliseconds. The default value is 200 (milliseconds). Values can be set from 50 to 1000.

Using the **no** parameter before the command returns the watchdog time to the default value.

Setting the script data collection limit

This parameter sets the data collection limit for the script in bytes. The script data collection limit is set as shown.

```
ServerIronADX(config)# script-profile sp1
ServerIronADX(config-script-profile-sp1)# data-collection-limit 50000
```

Syntax: **[no] data-collection-limit** <*limit-size*>

The <*limit-size*> variable is the new limit for data collection that you want to set in bytes. The default value is 100000 (100 Kbytes). While you can set this variable to any value that fits within the size of the system memory, all memory limits set for scripts are subject to their impact on system performance.

Using the **no** parameter before the command returns the data collection limit to the default value.

Setting the script restart limit

This parameter sets the maximum number of times that the script will restart after running into problems. Once the restart limit is reached, a syslog message is generated and the script is placed into a "suspended" state. Once in the "suspended" state the script can only be run if it is unbinded and rebinded or if the ServerIron ADX is restarted.

The script restart limit is set as shown.

```
ServerIronADX(config)# script-profile sp1
ServerIronADX(config-script-profile-sp1)# restart-limit 10
```

Syntax: **[no] restart-limit** <*limit-number*>

The <*limit-number*> variable is the maximum number of times that the script will restart after running into problems. By default a script will restart every time it is halted. This value can be set to any integer.

Using the **no** parameter before the command returns the restart limit to the default value.

Enabling script debugging

This parameter enables script debugging. With this option enabled, debug information is printed at the console. Script debugging is enabled as shown.

```
ServerIronADX(config)# script-profile sp1
ServerIronADX(config-script-profile-sp1)# debug
```

Syntax: **[no] debug**

If you have enabled scripting, you can use the **no** parameter to disable script debugging.

Setting the script print output

This parameter directs output from a print command within a script to either the console, syslog or disables it. The script can be set to print to the syslog as shown.

```
ServerIronADX(config)# script-profile sp1
ServerIronADX(config-script-profile-sp1)# print-output syslog
```

Syntax: **[no] print-output { console | syslog | none }**

The **console** parameter sets the script to print its output to the console.This is the default state.

The **syslog** parameter sets the script to print its output to the syslog.

The **none** parameter disables the script from printing.

Using the **no** parameter before the command returns the script print output to the default value

Displaying script information

The following sections describe the OpenScript information that can be displayed on a ServerIron ADX.

Displaying Run-time information about one or all scripts

Run-time information about a script can be displayed as shown in the following:

```
ServerIronADX# show script myscript.pl detail vip1 http
Script myscript.pl
==============================================================================
Virtual Server:               vip1
Service-Port:                 http
Script State:                 ACTIVE
Last Updated:
Script Restart:               0

Total Connections:            61
Concurrent Connections:       0


Error Counters:


Hits Per Event:
        HTTP Request event:                        9585575
ServerIronADX#
```

Syntax: **show script** <script-name> [**detail** <virtual-server-name> <port>]

The <*script-name*> variable allows you to specify a particular script to display information for. If a script is not specified, a list of the scripts stored on the ServerIron ADX will be displayed.

The **program** parameter displays the contents of the specified script

The **detail** parameter displays additional statistics described in Table 4. You can customize the statistics displayed by specifying a virtual server <virtual-server-name> and <port>. If not specified <virtual-server-name> and <port> are not specified, statistics are displayed for all ports.

TABLE 4 Runtime script statistics

This field...	Displays...
Virtual Server	The virtual server name.
Service Port	The service port
Script State	The state of the script: Active, Suspended or UpdatePending
Last Updated	The date and time that the script was last updated.
Script Restart	
Total Connections	
Concurrent Connections	
Error Counters	
Hits Per Event	
HTTP Request event	

Displaying scripts profile information

You can display the current settings for a script profile as shown in the following:

```
ServerIronADX# show script profile sp1
Script  profile sp1
======================================================
        Memory limit:             2000000
        Memory high water mark:   80%
        Time out:                 1000ms
        Data collection limit:    50000
        Restart Limit:            None
        Debug:                    off
        Print Output:             syslog
```

Syntax: **show script-profile** <*script-profile-name*>

The <*script-profile-name*> variable specifies the script profile you want to display the current settings for.

TABLE 5 Runtime script statistics

This field...	Displays...
Memory limit	The memory limit configured for the script.
Memory high water mark	The memory high water mark configured for the script as a percentage.
Time out	The time out value configured for the script in milliseconds.
Data collection limit	The data collection limit configured for the script in bytes.
Restart Limit	The restart limit configured for the script.
Debug	The status of the debug flag configured for the script. Can be off or on.
Print Output	The print output setting configured for the script: can be console, syslog or none.

Chapter

Script Example

4

Overview

The following sections of this chapter describe the entire process of writing a script, copying it to the ServerIron ADX and binding it to a virtual server port

Use case

This script is created in this example is designed to perform the following action on any SSL or HTTP traffic:

- It there is no X-Forwarded-For header, an X-forwarded-For header is added with the client source IP address: e.g. 4.4.4.4
- If a X-Forwarded-For header exists, the source IP address 4.4.4.4 is appended. For example, existing X-forwarded-For address: 1.1.1.1, 2.2.2.2, 3.3.3.3 + appended source address: 4.4.4.4, results in new source address: 1.1.1.1, 2.2.2.2, 3.3.3.3, 4.4.4.4.

Script for use case

The following script implements the example described in "Use case".

Example 1:

```
use OS_HTTP_REQUEST;
use OS_IP;
use OS_SLB;

BEGIN {
   $group1 = 10;
}
sub HTTP_REQUEST{
  $x_forward = OS_HTTP_REQUEST::header("X-Forwarded-For");
  $src_addr = OS_IP::src;

  if( defined $x_forward){
          $x_forward = $x_forward.", ".$src_addr;
          OS_HTTP_REQUEST::header("X-Forwarded-For", $x_forward);
  }else{
          OS_HTTP_REQUEST::push_header("X-Forwarded-For", $src_addr);
  }
  OS_SLB::forward($group1);
}
```

Copying and binding the script

The following command copies the script from a TFTP server to the to the ServerIron ADX.

```
ServerIronADX# copy tftp usb0 1.1.1.1 addip.pl sys\dapscript\addip.pl
```

The script is bound to the “vs1” virtual server as shown.

```
ServerIronADX(config)# server virtual vs1
ServerIronADX(config-vs-vs1)# port http script addip.pl
```

Sample ServerIron ADX configuration for use case.

The following is an example of a ServerIron ADX configuration which includes the required bind commands for the “addip.pl” script.

Example 1:

```
ServerIronADX# configure terminal
ServerIronADX(config)# server real rs1 100.1.5.113
ServerIronADX(config-rs-rs1)# port http
ServerIronADX(config-rs-rs1)# port http url HEAD/
ServerIronADX(config-rs-rs1)# port http group-id 10 10
ServerIronADX(config-rs-rs1)# exit
ServerIronADX(config)#
ServerIronADX(config)# server real rs2 100.1.5.114
ServerIronADX(config-rs-rs2)# port http
ServerIronADX(config-rs-rs2)# port http url “HEAD/”
ServerIronADX(config-rs-rs2)# port http group-id 10 10
ServerIronADX(config)# server virtual vs1 100.1.5.101
ServerIronADX(config-vs-vs1)# port http
ServerIronADX(config-vs-vs1)# port http script addip.pl
ServerIronADX(config-vs-vs1)# bind http rs1 http rs2 http
```