SHADOW OS/390 WEB SERVER™

USER'S GUIDE

NEON SYSTEMS, INC.

This document is published by the NEON Systems, Inc. Technical Publications Department and applies to Shadow OS/390 Web Server, Version *4*, Release *5*.

Address inquiries to:

NEON Systems, Inc.
14100 SW Freeway, Suite 500
Sugar Land, Texas 77478

World Wide Web: http://www.neonsys.com

Phone: 1-800-505-6366
(281) 491-4200 (Corporate Sales, Technical Support)
Fax: (281) 242-3880

December 3, 1999

Contents

About this Publication

This guide expands on the concepts and information presented in *Shadow OS/390 Web Server's Getting Started Guide*. If you do not find the information in this manual, refer to the *Getting Started Guide* or one of the related publications in Shadow OS/390 Web Server's set of manuals.

How this Publication is Organized

This book contains the following chapters:

- Chapter 1, "An Overview," provides an introduction to Shadow OS/390 Web Server by reviewing information covered in the Getting Started Guide plus additional information on rescanning to a new URL, error recovery, flush requests, and more in-depth information on HTTP and TCP/IP.
- Chapter 2, "The Shadow Event Facility (SEF)," covers the structure of an event procedure (header statements and process sections), the different types of variables, and how to control SEF from a batch environment.
- Chapter 3, "Defining Event Procedure Types," covers the different event procedures types, what they do, how they work and valid syntax for each.
- Chapter 4, "Web Transaction Security," covers security parameters and subsystem security.
- Chapter 5, "Writing Web Transactions in REXX," covers the syntax for coding REXX process sections.
- Chapter 6, "File Serving Using Shadow OS/390 Web Server," discusses supported files, how Shadow OS/390 Web Server handles files, and how to build WWW rules using /*FILE.
- Chapter 7, "HTML Extension Facility," covers the rules for coding HTML extension statements, run-time condition checking, iteration statements, and merge processing.
- Chapter 8, "Automated State Management Facility (ASMF)," covers State Information, the ability to 'remember' information at the end of a client/ server interaction because it has some bearing on a future anticipated interaction.
- Chapter 9, "Executing User Programs," discusses the program process section (what programs can be executed, where they must reside and how to code the process section), using other REXX Interpreters, and writing transaction programs (C/370, COBOL, and PL/I).
- Chapter 10, "Writing DB2-Based Web Applications," covers the operation of, coding and SQL statements for /*EXECSQL process section. The Shadow/

REXXTOOLs DB2/SQL interface is discussed in the HTML online documentation and the *Shadow Programming Guide*.

- Chapter 11, "Using TSO/E Services For Web Transaction Processing," discusses how TSO/E auxiliary servers operate, how to build the Shadow/REXX based TSO/E Web transaction, and the syntax and coding for the /*TSOSRV process section.
- Chapter 12, "AutoHTML - Web Enabling Transactions," discusses how to format the /*EXECIMS section. The *Installation Guide* covers how to configure the system so you can use AutoHTML.
- Chapter 13, "Data Mapping Facility," discusses the data mapping facility, how it works, the various ISPF panels and what it does.
- Chapter 14, "Shadow ADABAS Server," covers the new add-on component to NEON's line of Shadow Server products, which provide a reliable, high-performance access to ADABAS data from the desktop.
- Chapter 15, "Shadow_VSAM and Shadow_VSAM for CICS," covers the new add-on components to NEON's line of Shadow Server products, providing reliable, high-performance access to VSAM data.
- Chapter 16, "Shadow Web Interface," covers the GUI (Graphical User Interface) that allows you to perform many of the same functions available on the ISPF panels.
- Chapter 17, "Using the OS/390 UNIX OpenEdition Hierarchical File System (HFS)," covers support for the OS/390 Unix System Services Hierarchical File System (HFS).
- Appendix A, "Trace Browse," covers the different features of Trace Browse, such as, how it works, change columns (or displaying extra columns), and locating messages.
- Appendix B, "Trace Browse Archival Facility," discusses what it is, how it works, handling backups and extracts and configuring automatic backups.
- Appendix C, "Starting a Test Version," covers usinging the debugging control screen, setting the server up to run under TSO and using the code/370 debug tool.
- Appendix D, "Sever Error Codes," lists the different server error codes and their descriptions.
- Appendix E, "Supported SMF Fields," explains the records that are written out by Shadow OS/390 Web Server whenever a URL is executed. (Offset, Field Name, Field Type/Value, and Description)
- Appendix F, "Language Codes," lists the country codes and the language.
- "Glossary," lists and defines terms and acronyms that may appear in NEON Systems, Inc. publications.

Conventions

This book contains the following highlighting conventions:

BOLD CAPS
Identifies commands. For example:

Use the **KEYS** command to ...

`Monospace`
Identifies code examples, screen prompts, and messages, as well as directory paths. For example:

```
//STEP010    EXEC  PGM=NDBA2400
```

`Monospace Italics`
Identifies information you must provide at a screen prompt or in a text field. For example:

```
PARM='PARMLIB=your.parmlib'
```

<KEY> Identifies the key to press. For example:

<ENTER>

NEON Systems, Inc. uses *Release.Version* to identify software packages. For example, *Product 4.1*, denotes the fourth release, first revision of the software.

Reader's Comments

At NEON Systems, Inc. we are always looking for good ideas. If you have any comments or suggestions regarding any of our publications, please complete the Reader's Comment form (located at the back of this book) and return it to NEON, Attention: Technical Publications Department.

Mailing Address: **NEON Systems, Inc.**
14100 SW Freeway, Suite 500
Sugar Land, Texas 77478

Fax Number: (281) 242-3880

You can also send comments to directly to our Technical Publications department via the following email address: **documentation@neonsys.com**.

Thank you!

NEON Systems, Inc. Products and Publications

For a comprehensive list of the products currently marketed by NEON Systems, Inc., (*NEON*) visit our World Wide Web site at: **http://www.neonsys.com**.

You can also access and download all of the current NEON publications from this Web site.

Year 2000 Compliancy Statement

The following products from NEON Systems, Inc., are Year 2000 ready:

- **Enterprise Security Management Products**
- **Enterprise Subsystem Management Product Family**
- **Shadow® Product Family and Add-On Components**

  The mainframe code for the Shadow Product Family, Version 3.1 and all subsequent versions, are Y2K ready.

  All versions of the client code associated with Shadow® Direct™ and Shadow Enterprise Direct® are Y2K ready.

  > ***Note:***
  > While Shadow Direct, Shadow® OS/390 Web Server™, and Shadow Enterprise Direct are Y2K ready, customers should be aware that these products can provide access to data sources that may not be Y2K ready.

These products use four-digit year values both internally and externally (although, in a few cases, two-digit year values are displayed while four-digit year values are maintained internally).

Working with Technical Support

NEON Systems, Inc. provides a number of ways for you to obtain assistance for our products. All product support inquiries are handled by the same support group, regardless if you are a trial or a licensed customer. The following are available support options:

Support Option	How to Access	How it Works	This Option is Best for:
Email	Access to Technical Support via email: `support@neonsys.com` Email is available for receipt 24 hours a day, 7 days a week and is answered between 9AM-7PM CST Monday through Friday.	Email goes to the support queue, which is continuously monitored by a staff of cross-functional technical experts. It is answered in the order it is received. It is logged in the support database and assigned a trouble ticket number for tracking purposes.	This type of support is excellent for low to medium priority requests. It is a proven method for providing further information on critical problems that may have been phoned in. Email is a convenient way of sending us a list of lower priority items you have collected at a time that is convenient for you.
Phone	For access to Technical Support via phone, please call: **1-800-505-6366**	During normal working hours you will be transferred to someone who can usually answer your question on the first call. You may be required to page a support person via our phone mail system after hours.	This type of support is best for high priority requests and initial installation questions. Use this option for any obvious system errors or anytime you need the most rapid reply to your question.
Internet	For access to Internet support, please visit our website at: `www.neonsys.com`	Simply visit our website. NEON Systems works to keep current, relevant materials on our website to support our trial and licensed customers.	This option provides immediate access to documentation, updated client-side drivers, and our product Knowledge Base. The Knowledge Base is a collection of questions answered by support. Use this option to answer your own questions or to get a better understanding of what customers ask on an ongoing basis.
Account Manager	Call your NEON Systems Sales Representative at: **1-800-505-6366**	Your Sales Representative is your account manager. This person is ultimately responsible for your complete satisfaction with NEON Systems, Inc.	Contact your Sales Representative for pricing information, contract details, password renewal or if you feel your needs are not being met.

CHAPTER 1: An Overview

The *Shadow OS/390 User's Guide* expands on the concepts and information presented in *Shadow OS/390 Web Server's Getting Started Guide*. If you do not find the information in this guide, please refer to the *Getting Started Guide* or one of the related publications in Shadow Web Server's set of manuals.

What Is Shadow OS/390 Web Server?

Shadow OS/390 Web Server is a native MVS Web server which provides controlled access to MVS data and applications using any Web browser, such as Netscape Navigator or Internet Explorer. The Web Server does not require an intermediate server, nor is it limited to simple file transfers or screen scraping. Shadow Web Server is an MVS transaction processor product designed especially to connect MVS resident resources to the Internet or World Wide Web (WWW).

Internet Protocols*

The Internet operates effectively because an agreed upon set of communication rules and procedures are implemented consistently across all connected computers.

What Protocols Govern

A small subset of the things governed by these rules and protocols are:

- How data is transmitted and routed through the physical network pathways
- How remote computers, at previously unknown locations, are located, identified, and contacted
- When, and at what network junctures, data streams are validity checked
- Whether messages are discarded or retransmitted if an error is detected or a message cannot be delivered
- Whether one way, broadcast, or bidirectional sessions are used for transmission
- How messages are formatted by the sender
- How messages are interpreted by both the intended recipient and any intermediate computers along the transmission path

*See the *Getting Started Guide* for more information.

- Whether there is a client/server store and forward, or another relationship between communication partners
- Which message and session level responsibilities and liberties each transmission partner must observe

TCP/IP

TCP/IP is a communications protocol which provides many of the transmission, routing, error checking, and session establishment/breakdown services. TCP/IP also provides an easy to implement, extensible platform on which various application layer protocols have been constructed.

Internet Application Layer Protocols

There are many application layer protocols which have been built upon the TCP/IP communication protocol foundation. For example:

Domain Name Services (DNS)
These are used to look up computer domain names, their corresponding Internet Protocol (IP) addresses, and various other pieces of navigation and routing information.

File Transfer Protocol (FTP)
These are used to request and receive files and file system directory information from another computer.

HyperText Transfer Protocol (HTTP)
These allow the retrieval of virtually any digital file in a format suitable for later rendering in its original text, audio, or visual media presentation form on the Web.

Mail (mailto) This is used to transfer electronic e-mail messages.

Each application layer protocol defines:

- How communication protocol (TCP/IP) services must be used to establish communication sessions, make requests, send responses, recover from unexpected conditions and breakdowns in communication sessions.
- The message formats used between session partners, the roles of sender and receiver (such as client/server), and how senders and recipients are expected to process messages to provide useful services.

Client/Server Roles in HTTP

Many application layer protocols define communication partners as acting in client/server roles. This relationship also holds true for HTTP protocol. Each communications partner is either the client, which initiates a request, or the server, which responds to the request.

Terminology

Client	This refers to a Web browser, such as Netscape Navigator or Microsoft Internet Explorer and the person operating the browser. However, an HTTP client could be any software system that uses HTTP in a client role to initiate a request.
Server	This refers to Shadow OS/390 Web Server acting in an HTTP server role.

Note:

Formal HTTP specifications define a more generic term, "User Agent", used when referring to the downstream client or the upstream server. This is because an unknown number of intermediate proxy or firewalls servers can be between the target server and the original client.

Proxy servers and firewalls act as the server in relation to a downstream client, and as the client in relation to an upstream server.

Important HTTP Protocol Concepts

These are the most important concepts about HTTP protocol are:

Concept	Description
Single Request/Single Response	Only one request can be sent each time the client contacts a server and only a single response can be returned for each request. The server must send some response or close the session if it cannot. If the server does not respond in a reasonable amount of time, the client can close the session.
The Server Defines What Action is Taken	The client sends each request in a form that can look like a reference to a static data file, but it is the server that interprets the request. HTTP allows the server to perform virtually any action while generating a response to the client's request as long as the server obeys a few limitations imposed by the HTTP protocol. For example, HTTP allows the server to: • Execute existing CICS and IMS transactions and return results to Web browsers. • Execute SQL statements against DB2 and return results sets. • Execute programs in REXX, COBOL, PL/I, C, C++ and Assembler. • Retrieve MVS files or data sets. • Perform virtually any action while generating a response to the client's request.

Table 1–1. Important Protol Concepts

Concept	Description
Stateless Communication	Because each request/response interaction occurs in a separate communications session, each is logically isolated from all others. This makes HTTP a “stateless” protocol. There is no left over operational status information carried over from one request to another; it is “clean” or free from accidental informational artifacts. Various means are available both in the HTTP protocol and HTML specification to allow state information to be passed between client and server. However, it can also be difficult to create a coherent series of client/ server interactions using these means.
HTTP Messages Contain Self-Describing Information	The HTTP protocol defines self-describing request and response message formats for both inbound and outbound messages. For example, a client's request can indicate a text language preference, or only display GIF images. These controls allow the server to determine the best response to give each request. Server responses contain content type descriptions which tells the client how to present each response from the server. For example, the response could contain either an HTML page or plain text. The server can also use status codes in a response, such as requesting a userid and password, or redirecting the client to the server’s new location.

Table 1–1. Important Protol Concepts

Processing Web Transactions and URLs

An inbound URL (Uniform Resource Locator) specifies the location of a file on the Internet, network or hard drive; it also identifies the Internet service protocol, such as HTTP or FTP. Shadow OS/390 Web Server uses the URL Value to process transactions.

The following is a URL:

`http://www.neonsys.com/NEON/SAMPDATA/htxother.htm#attr`

Internet Protocol	Domain Name	Path	File Name	Bookmark
`http://`	`www.neonsys.com`	`/NEON/SAMPDATA`	`/htxother.htm`	`#attr`

Table 1–2. Parts of the URL

How the Web Server Handles URLs

The first part of the URL (`http://www.neonsys.com`) identifies the TCP/IP address of the MVS system to the server. Once the port has been reached, this information is discarded. Only the second part (`/NEON/SAMPDATA/htxother.htm#attr`) is passed to Shadow OS/390 Web Server for processing.

Discarded once it arrives at the port	SWS matches this to an Event Procedure Rule
`http://www.neonsys.com`	`/NEON/SAMPDATA/htxother.htm#attr`

Table 1–3. How Shadow OS/390 Web Server (SWS) Handles URLs

Handling Inbound Requests

Once Shadow OS/390 Web Server receives the URL, a Web transaction subtask is selected (spawned) to service the request. The subtask looks in a list of defined transactions to match the URL value against predefined WWW event procedure rule definitions. These definitions determine how the request is processed.

When a Match is Found

When the URL can be matched to a predefined WWW event procedure, the request is executed. Usually an outbound response is created where HTML or binary data is returned to the Web browser. Once the outbound response has been transmitted, the communications connection is terminated and the Web transaction subtask ends.

When No Match is Found

The Web Server performs an internal rescan.

Supported URL Values

Shadow OS/390 Web Server places restrictions on the URL and recommends that you limit the use of special characters when you create them to avoid any conflict.

Restrictions

Shadow OS/390 Web Server imposes the following restrictions on inbound URL values:

1. No URL string can be longer than 128 bytes in length.
2. The URL can contain any character except embedded blanks and a question mark.
3. If the transaction is defined outside of the master ruleset, the URL value must have a prefix value which relates the definition back to the dataset in which it is defined. This restriction is imposed to ensure that definitions are secure from unauthorized modification.

Special Characters and URL Strings

Special characters have accepted usage within HTTP and HTML specifications. Even though Shadow OS/390 Web Server does not restrict the use of these characters, we recommend that you limit their usage to avoid conflicts. If you do use these special characters, use them in the widely accepted Internet format. Failure to do so could cause display problems in Web browsers, or it could even confuse a proxy or routing agent, which in turn could misroute an inbound request.

Rules or Using Special Characters

Character	Reason Not to Use
embedded blank	This can never be used as part of a URL, because blanks are used as a delimiter within HTTP request and response messages.
question mark (?)	This can not be used, because it delimits the end of the URL string and signals the beginning of query variables within HTTP request messages.

Table 1–4. Avoid using these entirely

Character	Reason
pound sign (#)	This is used by most browsers to point to an HTML page's internal reference (bookmark). For instance, the HTML tag, `<A HREF="ABC.htm#xyz">` points to location xyz within the document at the URL location ABC.
Period (.)	This is normally used to signal the beginning of a file extension. Data following a single period can be interpreted as a PC or UNIX file extension, such as '.GIF', by the server. (See “Parsing URLs to Supply Missing /*FILE Keyword Values” on page 6-10 for more information.)
../ or ./	One or two periods, preceding a slash normally signal a relative URL value to both browsers and Internet proxy agents. Using this character combination can cause a proxy agent to misroute an inbound request.
Tilde (~)	This is used by many Web servers to denote a user owned and maintained directory or subdirectory path; a “private” Web site. This capability is currently not supported by Shadow OS/390 Web Server.
Exclamation point (!)	This will be used by a future enhancement to Shadow OS/390 Web Server.
Miscellaneous Other:	Many other symbols, particularly the ampersand (&), equal sign (=) and the HTML tag delimiters (< and >) can have special meaning to some proxy agents.

Table 1–5. Only use these in accepted Internet format

Rescanning to a New URL Value

When Shadow OS/390 Web Server detects an error during processing, such as an unmatched current URL or a security authorization error, it does not generate a hard coded error response. Instead, it performs an internal rescan request, which redirects processing of the client request to use a new URL-to-rule matching string.

Security related processing attributes accumulated during the previous URL-to-rule pattern matching are discarded and reaccumulated during the subsequent rescan.

Rescan Request

A rescan uses the newly specified URL string as though it were the URL originally requested by the client and initiates a new URL-to-rule match. It searches by beginning at the top most (least specific) WWW Rule with the default transaction run-time control attributes restored.

To reroute subsequent URL-to-rule matching activities in a WWW procedure before execution ends, set the RESCAN URL string value.

Error Recovery

Many of the error recovery processes are generated internally by a rescan, which redirects the processing to system supplied error recovery transaction definitions. These system recovery procedures are distributed within the WWW master ruleset.

FLUSH Request

A WWW rule procedure can explicitly request that:

- No further rule matches are to be made under any condition.
- The transaction should end immediately.

A FLUSH request causes unconditional termination of rule matching. Any buffered response data is transmitted to the client and the transaction ends.

Shadow/REXX Return Values

For **Shadow/REXX only**, flush or rescan processing is requested by setting a special RETURN value which is examined when the REXX procedure completes.

Return Value	Action
RETURN "FLUSH"	A Shadow/REXX procedure can end by returning this character string. The Web transaction ends and no further matching is performed against the inbound URL. Any data remaining in the output buffers is immediately transmitted to the Web client.
RETURN "RESCAN newURLstring"	A Shadow/REXX procedure can end by returning this character string. This value forces a transaction rescan to the new URL value. The “newURLstring” gives the new URL to be used in the next URL-to-rule matching lookup. This string must conform to the overall limits on WWW rule matching criterion.

Table 1–6. Special Shadow/REXX RETURN Values

Return Value	Action
Any other EXIT or RETURN value	This indicates a normal WWW rule procedure completion. Any processing of subsequent URL-to-rule matches and the final transmission of the outbound response data to the client proceeds normally.

Table 1–6. Special Shadow/REXX RETURN Values

Other WWW Transaction Procedures

All WWW transaction procedures types can request a subsequent rescan or flush operation using high level language SWSSET API call or invoke the REXX-language `SWSSET ()` function.

`SWSSET()` function calls must be issued before the user written program or REXX procedure ends. The final return code set at the end of a user written program execution or by other interpreters is always ignored unless an abort type error is indicated.

Recovery From Server Detected Errors

The Shadow OS/390 Web Server performs transaction level recovery operations in response to various error conditions. The error encountered and the action taken by the server is shown in the Table 1–7 on page 1-8.

Note:
Once a transaction procedure is initiated, it can encounter other authorization failures not described here. For example, a `System Abend S913` error could occur if the transaction procedure attempted to open an MVS dataset without a valid (authorized) run-time effective userid. Even though run-time authorization errors are intercepted by the Shadow OS/390 Web Server subsystem, they are reported back to the end user as “server errors” and not as authorization failures.

Transaction Level Recovery

Error Encountered	Action Taken by Shadow OS/390 Web Server
Communication session drops before entire HTTP transaction request is received.	Ends the transaction processing subtask and waits for new connection requests.
Inbound HTTP transaction request is badly formed. The structural failure may be the actual request sent inbound, or an attempt to use an HTTP/0.9 format request.	Rescans to “`SYSTEM/ERROR/400`” URL value to indicate HTTP request format error.

Table 1–7. Action Taken by Shadow OS/390 Web Server when an Error is Encountered

Error Encountered	Action Taken by Shadow OS/390 Web Server
An unsupported method is specified in an inbound URL sent from a Web browser.	Rescans to “`SYSTEM/ERROR/501`” URL value to indicate unsupported method to browser.
No match is found among any Web transaction definitions for the URL value requested by the client.	Rescans to “`SYSTEM/ERROR/400`” URL value to indicate URL Not Found.
A URL match is processed, but the transaction definition does not transmit any data to the Web browser before ending.	No action. The transaction ends. Most Web browsers display a message indicating that no data was received.
An end user userid/password is required by the transaction definition for authentication, but none was sent with the inbound request.	Rescans to “`SYSTEM/ERROR/401`” URL value to indicate the request is unauthorized. The realm value sent in the response is the server's subsystem ID value (for example, SWSS).
An end user userid/password is required by the transaction definition and it was supplied within the inbound transmission, but the userid/password authentication request is rejected by the MVS security subsystem. This includes all reasons for which a `RACROUTE REQUEST=VERIFY` operation might fail, except when the return code indicates an expired password.	Rescans to “`SYSTEM/ERROR/401`” URL value to indicate the request is unauthorized. The realm value sent in the response is the server's subsystem ID value (for example, SWSS)
Same as previous entry except the RACROUTE return code indicates the rejection is because the password associated with the userid has expired.	Rescans to “`PASSWORDEXPIRED`” URL value to begin a transaction/response dialog sequence which allow the end user to specify a new password value for the MVS security subsystem. The supplied “`PASSWORDEXPIRED`” Web transaction definition supports Netscape 2.0 browsers, and may require customization to operate correctly with other Web browser programs.
The userid is not authorized to execute the URL requested. This occurs when the generalized resource rule verification is used within the server for restricting users to specific URL values, and the userid does not have read authority to the generalized security subsystem resource rule.	Rescans to “`SYSTEM/ERROR/403`” URL value to indicate that the requested URL is forbidden. A link within the supplied `SYSTEM/ERROR/403` HTML points to the supplied `/RELOGON` transaction definition. The supplied “`/RELOGON`” Web transaction definition supports Netscape 2.0 browsers, and may require customization to operate correctly with other Web browser programs.
Shadow OS/390 Web Server detects an error within its own transaction processing routines. The possible conditions range from logic errors within the product coding to GETMAIN or other MVS system requests which cannot be honored by MVS.	Rescans to “`SYSTEM/ERROR/500`” URL value to indicate that a server error has occurred.
Shadow OS/390 Web Server detects an error or abend within a processing procedure executed as part of a matching Web transaction definition.	Rescans to “`SYSTEM/ERROR/500`” URL value to indicate that a server error has occurred.
The PLAN or SUBSYS keyword is specified on a `/*PROGRAM` section, but connection to DB2 fails for any reason.	Sets the WWW.AUXxxxxx variables to reflect the DB2 error condition which was encountered. Rescans to “`SYSTEM/ERROR/AUX`” URL value to indicate that a server error has occurred in an external component

Table 1–7. Action Taken by Shadow OS/390 Web Server when an Error is Encountered

Error Encountered	Action Taken by Shadow OS/390 Web Server
A DB2-related failure is encountered while processing a `/*EXECSQL` section. This includes problems with opening a DB2 connection or executing the SQL statement.	Sets the WWW.AUXxxxxx variables to reflect the DB2 error condition which was encountered. Rescans to "`SYSTEM/ERROR/AUX`" URL value to indicate that a server error has occurred in an external component.
A failure is encountered while processing a `/*TSOSRV` section. This includes problems with scheduling a TSO command processor or excessive CPU time usage by the command.	Sets the WWW.AUXxxxxx variables to reflect the TSO error condition which was encountered. Rescan to "`SYSTEM/ERROR/AUX`" URL value to indicate that a server error has occurred in an external component.
More than 25 rescan events have occurred while processing a single Web transaction.	Transmits hard coded error message to browser and end transaction.
Recursive error detected within a `SYSTEM/ERROR/nnn` procedure.	Rescans to `SYSTEM/ERROR/500` procedure to transmit a server error message.

Table 1–7. Action Taken by Shadow OS/390 Web Server when an Error is Encountered

A Word About HyperText Transfer Protocol (HTTP)

The HTTP protocol is a very robust enhancement of the File Transfer Protocol (FTP) because it:

- Transfers data files and other message data streams.
- Relays information describing the contents of each message.

File Transfer Protocol (FTP)	HyperText Transfer Protocol (HTTP)
A client establishes communications session with a server.	A client establishes a communications session with a server.
The client can request a directory change or display a list of files in the directory.	The client sends one, and only one request, for what appears to be a data file name to the server.
The client requests the download of one or more data files.	The server can retrieve an actual data file named in the request or it can perform a different action when servicing the request.
If possible and authorized, the server returns each requested data file to the client.	The sole obligation of the server is to return a response to the client's request.
The FTP server does nothing except package and transmit the requested files. It does not give the client any clues to the actual contents.	HTTP defines a robust message format which allows the server to return not only the response, but additional information describing the response. The client can examine this information to determine a response.
The server checks to ensure that the client received the file properly and may initiate retry processing.	The server does not verify that the response was correctly received.
An FTP client does nothing with the data files returned except to store them in a local file.	The client uses the additional information transmitted to determine how the response should be displayed.

Table 1–8. Comparison of HTTP and FTP Protocols

File Transfer Protocol (FTP)	HyperText Transfer Protocol (HTTP)
The communication session ends when either the client requests termination or the server terminates it after an inactive timeout expiration. There could be several interactions between client and server before the session is terminated.	The communications session is broken down, by the server, after each response message has been transmitted.

Table 1–8. Comparison of HTTP and FTP Protocols

TCP/IP Architecture

TCP/IP has four communication layers, which enable heterogeneous systems to communicate by performing network related processing, such as message routing, network control, and error detection and correction.

Application Layer

The appplication layer is provided by the program using TCP/IP for communication, such as, FTP, e-mail, or gopher. The interface between the application layer and the transport layer is defined by port numbers and sockets. See the *Shadow OS/390 Web Server's Getting Started Guide* for more information on the application layer.

Transport Layer

The transport layer provides communication between application programs wlhich are on the same or different hosts and can support multiple applications simultaneously. This layer is responsible for providing a reliable exchange of information. The main transport layer is TCP.

Ports and Sockets

Each process that communicates with another process identifies itself to TCP/IP by one or more ports. A port is a 16-bit number used by the host-to-host protocol to identify which higher level protocol or application program (process) to which it must deliver incoming messages.

Some higher level programs are themselves protocols. For example, TCP/IP standardized Telnet and FTP port numbers for all TCP/IP implementations. (Port 12 is used by a Telnet server; ports 20 and 21 are used by an FTP server.) These assigned port numbers are called “well-known ports”, while standard applications are called “well-known services”.

The well-known ports (numbers ranging from 0-1023) are controlled and assigned by IANA (International Assigned Numbers Authority). On most systems these ports can only be used by system processes or programs executed by privileged users. The ports ranging from 1024-65535 are not controlled and can usually be used by ordinary user developed programs.

Tip:
To avoid two different applications trying to use the same port numbers on one host, write your applications so they request any available port. This allows the port number to be dynamically assigned with different ports being used from one invocation of an application to the next.

Terminology

Socket This is a special type of file handle that a process uses to request network services for the operating system.

Socket Address
It consists of: {`protocol, local-address, port number`}

TCP/IP example: {`tcp, 38.158.124.40, 80`}

Conversation
This is the communication link between two processes.

Association
This completely specifies the two processes that constitute a connection. For example:
{`protocol, local-address, local-port, foreign-address, foreign-port`}

TCP/IP example:
{`tcp, 38.158.124.40, 80, 204.71.200.66, 21`}

Half-association
This would be either {`protocol, local-address, local port`} or {`protocol, foreign-address, foreign-port`}. It specifies half of a connection.

A half-association is also called a “socket” or a “transport address”. This means, the socket is an end point for communication which can be named and addressed in a network.

The socket interface is one of several application programming interfaces (APIs) to the communication protocols and was designed to be less a generic communication programming interface.

Internet Layer

This layer provides communication between computers and ensures that messages are delivered to the correct destination. This method is unreliable when it comes to connectionless packet delivery. Sent packets can be lost, out of order, or even duplicated. It is up to the higher layer protocols to deal with these problems.

One reason for using a connectionless network protocol is to minimize the dependency on specific computing centers which use hierarchical connection oriented networks. The Department of Defense's intent was to deploy a network that would continue to operate even if parts of the country were destroyed.

IP Addresses

An IP address consists of two logical parts:

- A network address
- A host address

The IP address is used by TCP/IP to route information packets from one node in the network to another. In order for the client request and the Web server's return response to get "delivered" to the correct computers, each computer on the Internet must have a unique numerical IP address assigned. This address is a series of four groups of numbers (ranging from 0 to 255), which are separated by periods. For example, `38.158.124.40`

You can refer to a Web site by its IP address (`http://38.158.124.40`) or by its name (`http://www.neonsys.com`).

Subnets

With the explosive growth of the Internet, a single IP address could no longer handle changes to local network configurations, such as:

- Adding a new type of physical network.
- Splitting the local network into two or more separate networks.
- Splitting networks into smaller networks with gateways between them.

To avoid multiple IP addresses, the concept of subnets was introduced. The assignment of subnets can be done locally, while the whole network still appears to be one IP network to the outside world.

Network Interface Layer

This layer, sometimes referred to as the link layer, data link layer, or network layer, is implemented by the physical network which connects the computers. Although this layer is not covered by the TCP/IP standards, there are specific methods used to access higher layers. The following are the different types of networks which the Internet layer can be connected.

Multiaccess broadcast networks

Here, any system (TCP/IP host) can have multiple connections to other hosts simultaneously, plus it can send information to all other hosts on the same network with a single message command. LANs are representative these networks. Protocols such as ARP, ProxyARP, RARP, BooP and DHCP are used with it.

Multiaccess nonbroadcast networks

Again, any host can have multiple connections to other host simultaneously; however, there is not a single messaging command that communication with all the hosts simultaneously. Examples of this type of network are X.25, Frame Relay, and AnyNet Sockets over SNA.

Point-to-point networks

Here, a host can only have one connection to one other host at a time. There are no broadcasting capabilities. Examples of this type of network are SNAlink and asynchronous connections (using SLIP or PPP).

Note:

The term connection applies to any single IP interface of a host in any of the network type. For instance, a host could have multiple point-to-point interfaces and thus more than one connection at a time, but still only one per interface.

Hardware Address Resolution

The Address Resolution Protocol (ARP) maps internet addresses to hardware addresses. When an application sends data over a TCP/IP network capable of broadcasting, IP requests the appropriate hardware address mapping using ARP. If the mapping is not in the mapping table (ARP cache), an ARP broadcast packet is sent to all hosts on the network, requesting their physical hardware address.

Because the asynchronous transfer mode (ATM) technology does not work in this manner, every host has to register with an ARP server on initialization in order to map IP addresses to hardware addresses.

Some network hosts do not know their IP addresses when they are initialized, especially when they are booted from diskette. Reverse ARP (RARP) can be used if you have an RARP server on your network to implement it. For example, if a diskless workstation knows its hardware address, then it can send a message to RARP to determine it own IP address.

CHAPTER 2: The Shadow Event Facility (SEF)

This chapter expands on the concepts and information presented in *Shadow OS/390 Web Server's Getting Started Guide*, which includes an introduction to the Shadow Event Facility, event matching and how it works, event procedure execution, event procedure rulesets, naming conventions, start-up parameters, dataset format, enabling and disabling event procedures, and the structure of an event procedure.

What It Does

The Shadow Event Facility (SEF) is a built-in feature and the foundation of Shadow OS/390 Web Server's WWW rule mechanism. With it, administrators can tailor their system using high-level language routines.

How It Works

When an event occurs (Shadow OS/390 Web Server receives a URL), SEF builds an event matching argument string by stripping off the destination information from the URL. It uses this information to search through a list of enabled event procedure rules until a match is found. SEF then executes the matching event.

In addition to matching URL requests to WWW rules, SEF contains more generalized event-to-rule matching capability. This allows you to schedule custom tailored procedures in response to processing events encountered during server operation, such as "trigger" the server to respond to a real-time event. For example:

Authorization Rules (ATH)
These are used to customize Logon and Logoff operations, control access to the programmable facilities of the Shadow Diagnostic Facility (SDF), and limit access to various server owned resources.

Exception Rules (EXC)
These are used to customize the server's response and handling conditions, such as SQL failures or when resource usage limit is exceeded.

SQL Rules (SQL)
These are used to capture, modify, or disallow processing for any server-based dynamic SQL operation.

Time-Of-Day Rules (TOD)
These are used to implement server administration tasks, such as refreshing web accessible data file from an external source. TOD fires at a specific interval, date, or time.

Event Types

The Shadow Event Facility recognizes the following event types:

Event Type	Description
ATH	Authorization events are generated by the server when authorization processing must be done. The event procedure can either do the required processing or allow default ACF/2 or RACF processing to make the final decision.
CMD	Command events control client/server and web access to the mainframe by using SEF to manage the environments on the host.
DIS	Disable events are pseudo-events generated by the system whenever an event procedure is being disabled. This allows a procedure to be executed when it is disabled in order to perform termination processing.
ENA	Enable events are pseudo-events generated by the system whenever an event procedure is being enabled. This allows a procedure to be executed when it is enabled so it can perform initialization processing.
EXC	Exception events are generated by the system when some action must be taken. For example, a long-running data base transaction has exceeded a predefined time limit. The event procedure can decide what action to take in response to the event.
GLV	Global variable events are generated by the system whenever 1) the value of a global variable symbol is changed and 2) when the SEFGLVEVENTS start-up parameter is set to allow them.
TOD	Time-of-day events are generated by the system at specified times.
TYP	Language Types are not actually events, but rather, procedures which are an integral part of SEF. TYP allows you to prototype new process section types by building compiler/interpreter in REXX, which is then processed on behalf of WWW events procedures.
WWW	World Wide Web event are generated by the system whenever an HTTP transaction arrives. This is the primary means transaction processing is implemented in Shadow OS/390 Web Server.

Table 2–1. Event Types

Event Matching*

When an event occurs, SEF builds an event matching argument string based on the event type. SEF uses this string to search through the list of enabled event procedure rules until one (or more) matches are found between the event argument string and the event procedure “criterion”. When a match is found, SEF “triggers” or “fires” the matching event procedure to execute.

Least-to-Most Specific

Event procedure criterion values are allowed to contain a wildcard character. This means a single event can match more than one event procedure. Event procedures are matched to the event argument and executed, in order, from the least-to-most specific match.

*Refer to *Shadow OS/390 Web Server Getting Started Guide* for more information.

Event Procedure Execution[*]

Normally, you want event procedure rules to run in direct response to an actual event; occasionally, you may need to specify that a procedure be allowed to initialize or terminate itself. Whenever a procedure is scheduled for execution, a pre-instantiated variable, PHASE, contains information about whether the procedure was invoked for initialization, termination, or normal event processing.

Event Procedure Rulesets

Event procedure datasets (rulesets) are MVS PDS datasets. Each member of the ruleset is either active (enabled) or inactive (disabled) and can be either:

- One event procedure rule.
- A REXX-language subroutine, which can be called by one or more event procedures contained within the same PDS ruleset.

Naming Convention

All event procedure dataset names must use the following Shadow OS/390 Web Server naming convention. All datasets must:

- Start with the same “prefix” (for example, '`SWS.xxxxxx.`'[†]).
- End with the same “suffix” (for example, '`.EXEC`').
- Contain a maximum of one qualifier between the pre-defined prefix and suffix. This mid-level qualifier is the “ruleset” or “event procedure set” name.

Start-up Parameters

Shadow OS/390 Web Server's startup parameters specify which MVS data set prefix and suffix values to use. The Shadow Events Facility then scans the MVS catalog and preloads enabled event procedures sets into storage. When a request is matched to an event, the SEF executes the matching event and the designated rule definition.

Event Procedure Dataset Format[‡]

Each event procedure ruleset contains one or more PDS members which must be allocated to contain either fixed or variable length records. We recommend you use numbered, RECFM(FB) datasets.

*Refer to *Shadow OS/390 Web Server Getting Started Guide* for more information.
†Where xxxxxx represents the version number. For example, `‘SWS.SV040100.’` is release 4.1 and `‘SWS.SV040500.’` would be release 4.5.
‡Refer to *Shadow OS/390 Web Server Getting Started Guide* for more information.

Enabling Event Procedures

Each member within an event procedure ruleset can be enabled and matched to the event type specified in the event procedure header. Enabled members are read into storage, compiled into an internal form, and then stored so they can be paired to event matching criterion values.

Note:
Only procedures that are enabled can be matched. Disabled procedures are ignored.

Enabling and Disabling Event Procedure Rules

A single event procedure rule can be enabled or disabled by:

- Using product ISPF panels.
- Flagging the event procedure with the PDS as “auto-enabled”.

If the member is auto-enabled, the Shadow Event Facility enables it at start-up. If it is not auto-enabled, the rule must be explicitly enabled using the ISPF interface after start-up.

Memory of the auto-enablement attribute is maintained across subsystem start-ups through the use of a bit within the PDS ISPF statistics. If you edit the member off-line, the auto-enablement is reset and is no longer auto-enabled.

Structure of an Event Procedure*

With the exception of WWW event procedures, each member in the PDS event procedure ruleset consists of two parts:

- A procedure header statement.
- One process section, which must be REXX. (The member cannot be enabled without it.)

The WWW event procedure can contain:

- Header-only rules (no process section).
- Process sections using REXX, FILE, Shadow/REXX and PROGRAM.

The following is an example of a WWW event procedure containing a valid header statement, a process section, and section contents:

```
/*WWW /NEON/HTMLFILE/* AUTHREQ(NO)
/*FILE DATATYPE(PDS) DDNAME(HTMFILE)
```

*Refer to *Shadow OS/390 Web Server Getting Started Guide’s* Chapter entitled, “Putting it All Together” for detailed examples of event procedures.

```
CONTENTTYPE(text/html)-
FORMAT(BINARY)
```

Event Procedure Header Statement (Required)

Header statements must:

- Be the first statement within the PDS member. If the first statement is not valid, then the PDS member cannot be enabled nor can it be processed by SEF.
- Begin in the first input column (discounting record numbering) of the first record within the event procedure member. If a valid header statement is not present, the PDS member cannot be enabled as an event procedure.

The Format

```
/*xxx  criterion  ( keyword ( keyword... ) cont )
```

Where

/*xxx Must be the first five characters on the header statement line. "xxx" is one of the valid Shadow OS/390 Web Server event types:

- **ATH -** Authorization event procedures
- **CMD -** Command event procedures
- **EXC -** Exception event procedures
- **GLV -** Global Variable event procedures
- **TOD -** Time-of-day event procedures
- **TYP -** Language processing 'type' definition
- **WWW -** World Wide Web Transaction Definition

Criterion Specifies the pattern match value (the event triggering criteria) for processing the event procedure rule. For example: `/NEON/HTMLFILE/*`

The exact contents and meaning of the criterion string are defined separately for each event type. The maximum length for URL criterion values is 50 bytes for all event types except WWW, which allows 128 bytes.

Keyword (optional)
Use one or more keywords to define:

- Properties of the event procedure.
- Attributes to be used in processing of the event. At present, keywords are used for specifying WWW security attributes.

For example: `AUTHREQ(NO)`

Cont (optional)

Use a continuation character after the header line keyword when the header statement continues on the next line. Two continuation characters are recognized:

- "-" This strips trailing blanks from the end of the continued line and from the beginning of the continuation line.
- "+" This interprets the string literally (blanks are not stripped).

Closing delimiter (optional)

If you use a closing "`*/`" delimiter at the end of the header statement and it follows all other keywords, the system ignores it. No information can follow the closing "`*/`" delimiter.

Process Section Header Statements

Except for WWW event procedures where header only rules are allowed, there must be at least one process section present within each event procedure member. See the *Getting Started Guide* for more information on process section headers.

Header-Only Rules

See the *Getting Started Guide*.

SEF Event Procedure Variables

By referencing the variable name within the Shadow/REXX (`/*REXX`) procedure, SEF event procedure variables are always accessible to a Shadow/REXX (`/*REXX`) process section without requiring special interface function calls to access or set the value of an SEF variable.

WWW Event Procedures

Because other non-REXX section types can execute a high-level language program in WWW event procedures, SEF event variables are accessible only under controlled circumstances. This usually requires the use of an access API function, such as the SWSVALUE function.

Types of Event Procedure Variables

Event Procedures use the following types of variables:

- REXX Dynamic Variables
- Event Related Variables
- Global Variables
- GLVEVENT Temporary Variables

REXX Dynamic Variables

REXX Dynamic Variables are variables that get created during execution of a REXX-language process section whenever you reference or set the value of a variable.

How They Work

REXX Dynamic Variables exist only during execution of a REXX-language procedure and are freed when the REXX environment is deleted. Generally, they are not accessed by any non-REXX procedure or function, except as explicitly noted in API functions.

Example:

In this example, a REXX-language simple variable, `COUNT` or `I` sets the value of the variable within the REXX code. The REXX-language compound variable symbol, `stemvar.I` sets the `“InitValue”`.

```
do I = 1 to COUNT
        stemvar.I = "InitValue"
     end
```

Not all Variable Symbols are Dynamic

Within a Shadow/REXX procedure, not all compound variable symbols are dynamic.

- If a compound variable has one of the stem values listed in the Global Variables section, it is a Global Variable.
- If a compound variable has the `GLVEVENT.` prefix, it is a GLVEVENT temporary variable.

Intercepting and performing special processing for these reserved stem values is an automatic feature of Shadow/REXX.

Global Variables

Global variables are special variables which Shadow OS/390 Web Server stores global variable checkpoint dataset. The values are persistent across restarts of the product and are shared by all event procedures which execute within the system.

Global variables begin with one of the stem values:

- `GLOBAL.`
- `GLOBAL1.` through `GLOBAL9.`

How They Work

When a global variable is referenced by a Shadow/REXX-language procedure or by the Web transaction API SWSVALUE function, the subsystem retrieves the value from its checkpoint dataset. When a global variable is updated, its new value is saved. Whenever the value of a `GLOBAL.` or `GLOBALn.` variable is changed, a GLV event can be generated to intercept the change and perform additional processing.

Note:
GLV events are only generated when the subsystem parameter, SEFGLVEVENTS, is set to YES during start-up.

GLVEVENT Temporary Variables

GLVEVENT temporary variables:

- Are designed primarily for use by the high-level language routines that create and interrogate variables during execution.
- Begin with the stem value `GLVEVENT.`

How They Work

GLVEVENT temporary variables are similar to event related variables because they exist only for the duration of the event being processed and are deleted when the event is completed.

These variables can be created or accessed from non-Shadow/REXX procedures using the SWSVALUE function or referenced by name within a Shadow/REXX procedure.

Refer to Chapter 8, “Automated State Management Facility (ASMF),” for more information.

Event Related Variables

Event related variables are created by the Shadow Events Facility whenever an event occurs. They are used to pass information about the event to the event procedure that is matched to the event. For example, a variable named, `WWW.INPUTURL`, receives a copy of the inbound URL transmitted from the Web browser.

How They Work

Event related variables:

- Can be referenced directly by Shadow/REXX-language process sections.
- Are only accessible to non-Shadow/REXX procedures using the SWSVALUE Web Server API function.

Changing Event Related Variables

Some event related variable values can be altered during execution; however, most are read-only. If you change the value of an event related variable, it can affect how the event is subsequently processed by Shadow OS/390 Web Server.

Note:
You cannot create new event related variables.

Changes are Cumulative

The first rule that an event triggers gets the original event information; rules for the same event executing later get modified copies of this information. Even if a rule modifies an event related variable, all rules eligible to execute for an event still execute.

Because the values of read-only event related variables do not change, all rules that execute for a single event get the same event data.

The PHASE Variable

Event related variables are specific to the event type with one exception. SEF always creates the event related variable PHASE before any event procedure is invoked. The PHASE variable can have the following values:

PHASE Variable Value	Meaning
INIT	This value is set whenever the event procedure is executed as a result of being enabled. Execution of a REXX-language procedure during enablement is optional.
TERM	This value is set whenever the event procedure is executed as a result of being disabled. Execution of a REXX-language procedure during disablement is optional.
PROC	This value is set whenever the event procedure is executed as a result of being matched to an actual event. This is the normal execution of the procedure.

Table 2–2. Values for the Phase Variable

Other Event Related Variables

The remaining event related variable values are specific to the event type and are only set up when the procedure is processing an actual event (such as, `PHASE="PROC"`).

Refer to Chapter 3, “Defining Event Procedure Types,” for more information on event related variables and event types.

Event Procedure Return Values

While an actual event is processed, one or more event procedures are matched to the event and executed. As each event procedure completes, it can set a return value. For some event types, this return value is intercepted and used to control how subsequent events are processed.

How They Work

Not all event types use this control mechanism. Refer to Chapter 3, “Defining Event Procedure Types,” for more information.

Return Values for the Pseudo-Events

The enable (ENA) and disable (DIS) events are classified as pseudo-events because they:

- Are only optionally generated when an event procedure is enabled or disabled.
- Invoke the event procedure which is being enabled or disabled.

If an event procedure is invoked for enablement (the PHASE variable is set to 'INIT'), or disablement (the PHASE variable is set to 'TERM'), it might not return a value.

Value Returned	Meaning
None Returned	If no value is returned, enablement or disablement processing ends with the rule in the desired state.
"TRUE"	If the character string value "TRUE" is returned, enablement or disablement also proceeds.
"FALSE"	If the character string "FALSE" is returned, enablement or disablement is suppressed. For example: • If the procedure was invoked for enablement, the event procedure is not enabled. • If the procedure was invoked for disablement, the event procedure is not disabled. It remains enabled, unless the subsystem is terminating.

Table 2–3. Return Values for the Pseudo-Events

Accessing SEF Variables

This table shows when each class of SEF Event Variables can be accessed.

Access Type and Variable Type	From Shadow/REXX Event Procedure	From High-level Language Event Procedure	From Other REXX Event Procedure
Reading Value of REXX Dynamic Variables	Intrinsic	Not Allowed	Intrinsic
Writing Value of REXX Dynamic Variables	Intrinsic	Not Allowed	Intrinsic
Reading Value of Event Related Variables	Intrinsic	Using SWSVALUE API call	Using SWSVALUE function call
Writing Value of Event Related Variables	Allowed only for certain variables	Not Allowed	Not Allowed
Reading Value of Global Variables	By Reference or using SWSVALUE function	Using SWSVALUE API call	Using SWSVALUE function call
Writing Value of Global Variables	By Reference or using SWSVALUE function	Using SWSVALUE API call	Using SWSVALUE function call
Reading Value of GLVENENT Temporary Variables	By Reference or using SWSVALUE function	Using SWSVALUE API call	Using SWSVALUE function call
Writing Value of GLVEVENT Temporary Variables	By Reference or using SWSVALUE function	Using SWSVALUE API call	Using SWSVALUE function call

Table 2–4. When SEF Variables can be Accessed

Controlling SEF from a Batch Environment

The `'NEON.CNTL'` library contains the member, `'SWSBATCH'`, which illustrates two ways in which Shadow/REXX can be invoked:

- **Under a batch mode TMP (preferred).** SAY statements and TRACE output are intercepted and written to `SYSTSPRT`.
- **Stand-alone.** SAY statements and TRACE output are directed to the MVS console, which can be cumbersome if you're trying to write and debug a new procedure.

Sample REXX Routines

The sample REXX routines are named `'SDBBATCH'` and `'SWSBATCH'` in SYSEXEC. Both samples refer to a ruleset which does not exist at your site (`test.andy1`). The procedure in the sample illustrates:

- The use of all SEF commands that can be used to control rulesets.
- The format of other SEF commands.

Return Messages

Whenever rule enablement requests are issued via an SEF command, always retrieve the messages from the external data queue to find out the status of the operation. If the enable operation fails because of an invalid rule syntax, a message is issued. If no message is returned, it means the rule was successfully enabled. (This applies to the **ENABLE** and **START** command verbs).

Note:
A return code of ZERO does not indicate the final results of processing the command, rather, it indicates the command was successfully passed to the subsystem and that some response was received. The External Data Queue must be examined to determine the actual results which occurred while processing the command.

Return Values

Whenever an ADDRESS SEF command is issued, the following return code value is set for the REXX variable “RC”:

Return Code Value	Meaning
0	SEF COMMAND ISSUED AND RESPONSE RECEIVED
4	WARNING MESSAGE ISSUED
8	COMMAND TIME OUT ERROR
16	SEF COMMAND SYNTAX ERROR
20	TARGET SUBSYSTEM NOT ACTIVE
24	INCOMPATIBLE SUBSYSTEM VERSION
28	INTERNAL SENDMG ROUTINE FAILED
32	USER NOT AUTHORIZED TO ISSUE THIS SEF COMMAND
36	AUTHORIZATION PROCESSING ABENDED
40	HOST COMMAND FAILURE

Table 2–5. Return Values

CHAPTER 3:
Defining Event Procedure Types

This chapter expands on the concepts and information presented in *Shadow OS/390 Web Server's Getting Started*, which includes the structure of an event procedure, event procedure header statements and the format, process section header statements, process section header syntax, header-only rules, SEF event procedure variables.

Different Event Procedure Types

SEF recognizes the following event types:

- Authorization (ATH) events
- Command (CMD) events
- Disable (DIS) events
- Enable (ENA) events
- Exception (EXC) events
- Global Variable (GLV) events
- Time-of-day (TOD) events
- Language Types (TYP)
- Word Wide Web (WWW) events

Refer to the appropriate event types in this chapter for specifc information on:

- The format of the criterion value which must be present on each event procedure header statement
- Which keywords can be specified for each event procedure header statement
- Which process types can be created within the process sections
- What the valid return values are for the event type
- What valid event related variables are created for each each event type

Authorization (ATH) Event Procedures

Authorization (ATH) Event Procedures are generated by the server when authorization processing must be done. They can be used to:

- Tailor the operation of the server at security control points.
- Perform complex, site-specific processing in response to an access request.
- Can either do the required processing or allow default ACF/2 or RACF processing to make the final decision.

How They Work

Whenever the server performs authorization processing for a controlled resource, an ATH event procedure is scheduled for execution. The server:

1. Generates an ATH event criterion string which describes the resource being checked or the operation being performed before invoking the validation facilities of RACF, ACF/2, or TopSecret.
2. Searches the enabled ATH rules for a matching event procedure.
   a. If a match is found, the ATH rule is scheduled for execution before the server attempts to validate access using RACF, ACF/2, or TopSecret.
   b. If no matching ATH procedure is found, the server relies entirely upon the determination of the MVS security subsystem.

Each ATH procedure ends by returning one of the following signals:

- Unconditionally REJECTs access to the resource being check. The server bypasses invocation of RACF (ACF/2 or TopSecret) services and rejects the resource access request.
- Unconditionally ALLOWs access to the resource. The server bypasses RACF processing and immediately allows access to the resource.
- Makes no decision about access to the resource. In this case, the server proceeds by invoking the MVS security subsystem to determine if access to the resource is allowed or denied.

ATH Event Procedure Criterion

An ATH event procedure criterion value is a string containing from 1 to 30 characters. A trailing asterisk can be used as a wildcard character. If a single asterisk is coded as the criterion, the ATH rule is scheduled for all control points. For example:

```
/*ATH eventcriterion
```

Fixed Control Points

LOGON and LOGOFF ATH criterion are fixed. LOGON.WWW is the criterion value for HTTP client userid and password validation requests. LOGOFF.WWW is the criterion for client userid logoff processing requests. All third-party run-time userid logon and logoff processing is handled internally. ATH rules are not scheduled for non-client verification logon/logoff control points.

Other Control Points

All other server control point events are generated when an attempt is made to access a server provided or server controlled facility. Each control point has an associated “generalized resource name” used by the server. These names are fixed and are not user configurable.

The server constructs the ATH event matching string by placing a period following the fixed, generalized resource name. When an individual resource entity identity is known, the entity name is appended to the string following the period.

The “generalized resource names” referred to here are used to perform MVS security subsystem resource checking. Start-up parameters give the RACF “class table” name, in which these resource names must be defined. The MVS security subsystem is configured to grant or deny access to the resources, by name, within each class. The resource names in use by the server are normally configured into the MVS security subsystem at installation time. The resource names are:

Resource Name	Server Controlled Resource
CONTROLBLOCKS	View internal server control blocks using ISPF interface.
CICSCONNECTIONS	View or define server CICS connection entities.
FILE	View or control server file services ISPF interface.
DATABASES	View or define a database entity.
SEF	Access the Event Procedure Management functions of the server.
TSO	Schedule out-board execution of a procedure in a TSO server managed by the server.
LINKS	View or define a telecommunications link entity.
MQSERIES	View or define an MQ Series entity.
PARMS	View or set a server initialization parameter value.
RPC	Execute a user-written transaction program.
TRACEDATA	View binary data for each wrap-around trace entry.
TRACEBROWSE	Access the wrap-around trace display facility.
SWS	Access the server's ISPF interface.
TOKENS	View or control server Tokens.
USERS	View in-flight transaction display.
GLOBALS	Use ISPF interface to display/alter server Global Variables.
DATASET	Access a specific MVS dataset.
URL	Check authorization to execute a Web transaction definition which uses resource protection.

Table 3–1. Server Controlled Events

ATH Event Procedure Header Keywords

No keywords are currently defined for ATH event procedures. Only the event criterion value is allowed and required. Each ATH Header Statement is coded as follows:

```
/*ATH eventcriterion
```

Allowed Process Sections

A REXX-language process sections must be coded in each ATH event procedure. Header-only ATH rules are not allowed.

ATH Valid Return Values

When an ATH procedure ends, the return value set by the REXX language rule is examined by the server before invoking MVS security routines.

Return Value	Action
ACCEPT	If the REXX procedure returns the string **"**ACCEPT**"**, the server allows access to the requested resource and bypasses any further processing by the MVS security subsystem.
REJECT	If the REXX procedure returns the string **"**REJECT**"**, the server denies access to the requested resource and bypasses any further processing by the MVS security subsystem. The ATH rule may set the ATH.OPAUERMG variable to a message to explain the rejection. In most validation requests, this error message is forwarded to the requester.
Any other value	If any other value (or no value) is returned, the server performs validation checking using built-in MVS security subsystem interfaces. The final determination to allow or deny access is made by RACF (ACF/2 or TopSecret).

Table 3–2. Valid Authorization Return Values

ATH Event Procedure REXX Variables

For any ATH event which is scheduled on behalf of an in-flight Web transaction task, the server makes the transaction's WWW. stem variables available to the ATH procedure. These variables are not available to ATH rules scheduled outside a Web transaction subtasks. The WWW. variables that the server maintains can be found at WWW Event-Related Variables.

In addition to the WWW variables, each ATH procedure also has access to all server wide "`GLOBAL.`" variables.

ATH-specific variables are created before the ATH procedure is invoked. The exact variables created differ, depending on the type of resource check operation being performed. The variables which are instantiated for ATH procedures are based upon the resource type being checked.

The following tables show the ATH. stem variables built for each resource validation type.

ATH Stem Variable	Description
ATH.OPAURQSR	Contents: The resource name string is set to LOGON or LOGOFF for client authentication processing events. See “Fixed Control Points” on page 3-2. Data Type: Character, Read-only
ATH.OPAUACSR	Contents: The type of access being requested for the resource as a server defined string. The access type string varies depending on the resource being validated and the operation being requested. See Table 3–8 on page 3-7 for valid values. Data Type: Character, Read-only
ATH.OPAURQRC	Contents: This requests a return code. Valid values are: • 00 - Request Allowed • 04 - Request must be modified • 08 - Request Failed • 12 - Request Abended • 16 - Product Address Space is down Data Type: Character, Read-write
ATH.OPAUUSID	Contents: This is the userid being validated for LOGON, LOGOFF, or some task requesting access to the controlled resource. A LOGON routine may change the value of this field to cause a rule generated userid to be used for subsequent validation processing in RACF, ACF/2 or TopSecret. All other ATH rules should not attempt to alter this variable. Data Type: Character, Read-only except as noted.
ATH.OPAUERMG	Contents: This is a validation error message variable which can be set by the ATH procedure. Data Type: Character, Read-write
ATH.OPAUSRID	Contents: This is the full ATH event procedure criterion value set by the server for this control point. Data Type: Character, Read-only

Table 3–3. ATH. Variables Set for ALL Rules

ATH Stem Variable	Description
ATH.AUCCID	Contents: This is the CICS SYSIDNT name. Data Type: Character, Read-only

Table 3–4. ATH. Variables Set for CICSCONNECTIONS

ATH Stem Variable	Description
ATH.AUBKCBNA	Contents: This is the control block acronym. Data Type: Character, Read-only
ATH.AUBKCBAD	Contents: This is the control block virtual storage address. Data Type: Character, Read-only
ATH.AUBKCBLN	Contents: This is the control block length. Data Type: Numeric, Read-only
ATH.AUBKCBAS	Contents: This is the Address Space ID (ASID) of the control block. Data Type: Numeric, Read-only

Table 3–5. ATH. Variables Set for CONTROLBLOCKS

ATH Stem Variable	Description
ATH.AUDBNAME	Contents: This is the database name. Data Type: Character, Read-only
ATH.AUDBTYPE	Contents: This is the database type. Data Type: Character, Read-only
ATH.AUDBHOST	Contents: This is the database host name. Data Type: Numeric, Read-only

Table 3–6. ATH. Variables Set for DATABASE

ATH Stem Variable	Description
ATH.AUGLRQTY	Contents: Used to specify the global access request type: • A - Some type of read access • U - Some type of update access Data Type: Character, Read-only

Table 3–7. ATH. Variables Set for GLOBALS

ATH Stem Variable	Description
ATH.AUGLOPCH	Contents: Used to specify the operation subtype value: • A - Add a global variable. • D - Drop a global variable. • E - Check global variable existence. • F - Exist/obtain for global variable. • I - Obtain information about a global variable. • L - List information about a global variable. • O - Obtain a global variable. • R - Remove a global variable. • S - Subtree processing. • T - Subtree information processing. • U - Update a global variable. • V - Value Processing. Data Type: Character, Read-only
ATH.AUGLDELN	Contents: This is the global variable's derived name length. Data Type: Numeric, Read-only
ATH.AUGLDENA	Contents: This is the global variable's derived name. Data Type: Character, Read-only

Table 3–7. ATH. Variables Set for GLOBALS

ATH Access Type Values

The server generates an access type string value for each resource validation request. The string used depends on the resource being validated.

The following table gives the access type strings that the server sets:

Type String	RACF Value	Access Type Used For Validation Of
EXECUTE	Execute	RPC, URL
READ	Read	DATASET
INFO	Read	PARMS
LIST	Read	CICSCONNECTIONS, FILE, DATABASE, SEF, TSO, LINKS, MQSERIES, PARMS, TOKENS, USERS, GLOBALS
SHOW	Read	PARMS
DISPLAY	Read	CONTROLBLOCKS, CICSCONNECTIONS, FILE, DATABASE, SEF, TSO, LINKS, MQSERIES, PARMS, TOKENS, USERS, GLOBALS, TRACEDATA, TRACEBROWSE
SET	Update	PARMS

Table 3–8. ATH Access Type Values

Type String	RACF Value	Access Type Used For Validation Of
CONTROL	Control	DATASET, LINKS, SEF, FILE, DATABASE, CICSCONNECTIONS
KILL	Update	Users
WRITE	Update	CONTROLBLOCKS, CICSCONNECTIONS, FILE, DATABASE, SEF, TSO, LINKS, MQSERIES, PARMS, TOKENS, USERS, GLOBALS, TRACEDATA, TRACEBROWSE, DATASET
ADD	Add	CICSCONNECTIONS, DATABASE, LINKS, MQSERIES, GLOBALS
MODIFY	Update	CONTROLBLOCKS, CICSCONNECTIONS, FILE, DATABASE, SEF, TSO, LINKS, MQSERIES, PARMS, TOKENS, USERS, GLOBALS, TRACEDATA, TRACEBROWSE, DATASET
DEFINE	Add	CICSCONNECTIONS, DATABASE, LINKS, MQSERIES, FILE
CONTROL	Control	DATASET
DELETE	Delete	DATASET
ALTER	Alter	DATASET
USER	UserID Password Validation	LOGON
LOGOFF	UserID Logoff	LOGOFF

Table 3–8. ATH Access Type Values

Command (CMD) Rule Event Procedures

Command rules control client/server and Web access to the mainframe by using the Shadow Event Facility (SEF) to manage the environments on the host. When viewed in conjunction with diagnostics, monitoring and control mechanisms built into Shadow OS/390 Web Server, command rules are another step in providing comprehensive Automated System Operation (ASO) capabilities.

Note:
Command rules are not subject to any security. Because these rules can access and update any part of the product without constraint, each installation needs to control who can create, enable, and disable command rules.

How They Work

Whenever the server receives a command from an MVS console, a command rule event procedure is scheduled for execution. The console can be either 1) a physical console used by operations personnel or 2) extended software used by other products (such as NetView, CA-OPS/MVS, or SDSF.)

The command rule text consists of a command verb, followed by operands (optional). This verb string is matched against enabled command rules from the least to most specific. Command rules can:

- Examine the command text, parse out the operands and take whatever action is needed, such as read and set product parameters. (This means parameters can now be displayed and changed from an MVS console.)
- Access and update NEON REXX Global variables.
- Communicate back to the console that entered the command using REXX SAY statements. All SAY statement output is routed back to the console which entered the original command. This is not only useful to operators entering commands from consoles, but it also allows ASO products to communicate with, interrogate the status of, and control Shadow OS/390 Web Server as needed.

Command Rule Processing

All command rule processing is done using Shadow/REXX. Processing in TSO/E REXX, CA-OPMS/MVS REXX, or other programming languages is not supported at this time. When the Command (CMD) procedure ends, it returns one of these values:

- RETURN - Command rule has finished execution.
- RETURN "ACCEPT" - Command rule has finished execution.
- RETURN "REJECT" - Command rule has rejected execution of the command.

Command Rule Syntax

Command rules are triggered by either the MVS **STOP/MODIFY** interface or directly via an MVS command using the subsystem name chosen at startup time as the identifying parameter. For example:

```
SWSX command text  or  F SWSX, command text
MODIFY SWSX, command text
```

From the perspective of command rule processing, there is no difference between these methods.

Note:
The command rule event facility cannot be used for general MVS command processing. The command must be directed at a specific instance of the product identified by the subsystem name chosen at product initialization.

Event Procedure Criterion

A CMD rule event criterion is a string anywhere from 1 to 30 characters. A trailing asterisk can be used as a wildcard character. If a single asterisk is coded as the criterion, the CMD rule is scheduled for all commands. For example:

```
/*CMD eventcriterion
```

Rule Matching Order, Least to Most Specific

The event criterion can be specific or generic. The verb string is matched against all enabled command rules. Matched command rules are executed in order of least specific to most specific.

Example: Enabled command rules

/*CMD * Matches all commands.

/*CMD SE*
Matches all commands beginning with SE.

/*CMD SEF
Matches command name SEF only.

If the following command is issued:

```
SWSX SEF xxxxx or F SWSX,SEF xxxxx
```

First, Rule 1 is matched and executed, then rule 2, and finally rule 3.

Event Procedure Header Keywords

Currently, no keywords are defined for CMD event procedures. Only the event criterion value is allowed and required. Each CMD Header Statement is coded as follows:

```
/*CMD  eventcriterion
```

Allowed Process Sections

A REXX-language process section must be coded in each CMD event procedure. Header-only CMD rules are not allowed. To code the REXX process section header statement see `/*REXX` Process Sections.

Valid Return Values

When an CMD procedure ends, the return value set by the REXX language rule 1) is examined by the server and 2) controls what is done with the command.

Return Value	Action
None supplied	If the REXX procedure simply executes a RETURN command, the server sends a return code which indicates successful completion of the command rule. See Special Considerations for STOP rules.
ACCEPT	If the REXX procedure returns the string "ACCEPT", the server sends a return code which states the command rule was successfully completed. See Special Considerations for STOP rules with RETURN ACCEPTED.
REJECT	If the REXX procedure returns the string "REJECT", the server sends a return code which states the command rule rejected the entered command. It is the responsibility of the person implementing the command rule to state (using REXX SAY statements) the reason the command was REJECTED. See Special Considerations for STOP rules with RETURN REJECTED.

Table 3–9. CMD Valid Return Values

Special Considerations for STOP Rules

The MVS **STOP** command can also drive command rule processing. In turn, command rule processing can control or reject product shutdown.

Note:
The criterion of a **STOP** command rule must be **STOP** (or a less specific command rule that matches **STOP**). The MVS **STOP (P)** command also drives a command rule with a matching criterion of **STOP** but the MVS P command does not drive a command rule with a criterion of P.

The return value supplied by the **STOP** CMD rule determines product termination. Product shutdown can be controlled or rejected with **STOP** CMD rule processing. The valid return values and their functions in a **STOP** CMD rule are:

Return Value	Action
None supplied	Product termination is allowed to continue.
ACCEPT	Product termination is not allowed to continue.
REJECT	Product termination is not allowed to continue.

Table 3–10. Return Values Supplied by the STOP CMD Rule

CMD Event Procedure REXX Variables

Variable	Description
CMD.VERB	Contents: The command name entered at the console. Data Type: Character, Read-only
CMD.TEXT	Contents: Any operands entered after the command name at the console. Data Type: Character, Read-only

Table 3–11. CMD Variables Set for ALL Rules

Exception (EXC) Event Procedures

Exception (EXC) Events Procedures are generated by the system when an action must be taken. For example, a long-running data base transaction has exceeded a predefined time limit. The EXC event procedure decides what action to take in response to the event.

How They Work

The server schedules execution of enabled EXC procedures when certain exceptional events are detected. With EXC rules, you can customize various time-slicing and time-keeping facilities of the server.

EXC Event Procedure Criterion

The criterion value for EXC event procedures is server generated exception type names, which are shown in the following table:

Exception Name	Description	Default Server Action
CPULIMIT	A transaction task has exceeded its maximum CPU time limitation. This exception is detected only when multi-part messages are being transmitted, and only at the point when a new message segment is being read.	Terminate the transaction task.
CPUTIME	A transaction task has exceeded its maximum CPU time limitation. This exception can be detected at any time during execution of the transaction task.	Terminate the transaction task.
IMSFAIL	An IMS task detected a failing IMS operation. This exception can occur for any type of IMS processing.	Terminate the IMS operation and reflect error back to the client transaction task.
LOCKEXCLUSIVE	A transaction task has exceeded its DB2 exclusive lock time limit.	Terminate the transaction task.

Table 3–12. Exception Names

Exception Name	Description	Default Server Action
LOCKSHARE	A transaction task has exceeded its DB2 shared lock time limit.	Terminate the transaction task.
LOCKUPDATE	A transaction task has exceeded its DB2 update lock time limit.	Terminate the transaction task.
PERSQLCPU	A transaction task has exceeded its per SQL statement CPU time limit. This exception is only detected by SQL operations executed by the server (such as for EXECSQL rules) and not when a user written high-level language program invokes long running SQL operations.	Terminate the transaction task.
SQLFAIL	A transaction task detects an SQL statement has failed. An SQL statement is considered to have failed if a negative SQL code is set. This exception is only detected by SQL operations executed by the server (such as for EXECSQL rules) and not when a user written high-level language program invokes long running SQL operations.	Reflect SQL error code to the transaction task.
TIMERONLIMIT	A transaction task detects that a prepare has returned a timeron value which exceeds the tasks timeron limit value. This exception is only detected for dynamic SQL operations executed by the server (such as for EXECSQL rules) and not when a user written high-level language program invokes prepare.	Terminate SQL processing a reflect error to transaction task.
WAITTIME	A transaction task has exceeded its wait time limit. This exception can be detected at any time for transaction tasks.	Terminate the transaction task.

Table 3–12. Exception Names

EXC Event Procedure Header Keywords

No keywords are currently defined for EXC event procedures. Keywords can not follow the criterion value on the event procedure header statement.

EXC Allowed Process Sections

Only REXX-language process sections can be coded. Header-only rules are not allowed.

EXC Valid Return Values

When an EXC procedure does not return a value, the default action is taken by the server as seen in the event procedure criterion table.

Special Return Values

For some exception types, the EXC procedure can set a special return value which causes the server to take an alternate recovery action. For example, it could ignore a CPU time limit overrun or increase a task's CPU time limit value.

The run-time environment in which an exception is detected by the server varies for each type. Some exceptions are detected synchronously, within a transaction processing subtask, while others are detected asynchronously by a special server timekeeping task (the "check limits" task).

Samples

The EXC procedure samples distributed with the server contain a sample for each of the exception types. Instructions in the samples indicate the environment in which the exception is detected, what operational controls can be used to affect subsequent processing by the server, and what return values are valid.

EXC Event Procedure REXX Variables

The server causes the following REXX variables to be instantiated, which are available for examination by the EXC event procedure.

Variable Name	Description
EXC.OPEXSRID	Contents: This is the exception name which is matched to the EXC procedure's criterion value. Data Type: Character, Read-only
EXC.OPEXACSR	Contents: This is a string describing the default exception handling action which the server takes for this exception. Data Type: Character, Read-only
EXC.OPEXERMG	Contents: This is an error message which can be set/changed by the EXC event procedure. Data Type: Character, Read-write
EXC.OPEXCNTK	Contents: This is a connection token value, which can be used to by the EXC event procedure to obtain information about the transaction processing task. Note: The token must be used to access transaction task information in cases where the exception is detected asynchronously. Data Type: Character, Read-only
EXC.OPEXWAOK	Contents: This is a variable which indicates if the EXC procedure is allowed to perform operations which cause the current subtask to be placed in a wait state, for instance, by issuing an I/O request. The values set for this variable are: • 0 - Waits are not allowed • 1 - Waits are allowed Data Type: Character, Read-only

Variable Name	Description
EXC.OPEXINFO	Contents: This is a variable which indicates whether the SWSINFO function can be used by the EXC procedure. The values set for this variable are: • 0 - SWSINFO should not be used • 1 - SWSINFO can be used Data Type: Character, Read-only

Table 3–13. EXC Event Procedure REXX Variables

Global Variable (GLV) Event Procedures

Global Variable (GLV) Events Procedures are generated by the system whenever 1) the value of a global variable is updated and 2) the SEFGLVEVENTS start-up parameter is set to allow them. A GLV event procedure can be used to produce side-effects based upon the detection of an update to a global variable value.

How Global Variable (GLV) Work

When the update event is detected, the SEF attempts to locate a matching GLV event procedure.

Note:
The use of GLV event procedures has a modest impact upon the virtual storage utilization of the Shadow OS/390 Web Server subsystem.

GLV Event Procedure Criterion

You may specify a 1-to-50 byte string for the criterion value of the GLV event procedure header statement. If you specify the value in lowercase, SEF converts it internally to uppercase. Shadow OS/390 Web Server processes all GLV matching operations using uppercase.

The criterion value is the name of the global variable which must be altered for this event procedure to be triggered for execution. A generic procedure can be created by using the wildcard (`*`) character.

Example

The sample GLV procedure below would trigger for any update to a global variable which has a name beginning with "`GLOBAL.WEBTRANS.`".

```
/*GLV  Global.Webtrans.*
/*REXX
    .
    ...remainder of procedure
```

GLV Event Procedure Header Keywords

No keywords are supported for GLV event procedures. Keywords should not follow the criterion value on the event procedure header statement.

GLV Allowed Process Sections

Only REXX-language process sections can be coded. Header-only rules are not allowed.

GLV Valid Return Values

The Shadow Event Facility is unaffected by any values returned by a GLV event procedure. The Global variable value is updated in all cases, regardless of the return value set by a GLV procedure.

GLV Event Procedure REXX Variables

When a global variable change event is detected, the system extracts information about the event and creates event-related variables. These variables are pre-instantiated when the GLV event procedure is scheduled for execution.

You may access these data items directly, by name, from within REXX language event procedures.

Variable Name	Contents
PHASE	Contents: This contains a four-byte character constant which indicates the processing phase for which the current event procedure was invoked. • If set to "INIT", the procedure is enabled either during subsystem start-up or in response to a user enable request. • If set to "PROC", the procedure runs after being matched to an actual global variable update event. • If set to "TERM", the procedure is disabled either during subsystem shut-down, or in response to a user disable request. During procedure enablement and disablement, the only other variable that is instantiated is `GLV.USER`. The remaining variables are only instantiated during "PROC" phase processing. Data Type: Character, Read-only
GLV.NAME	Contents: This is 1-to-50 byte derived name of the global variable whose modification triggered this event. Data Type: Character, Read-only
GLV.NEWVALUE	Contents: This is the global variable's value after modification. Note that the standard REXX definitions apply to variables that have never been referenced before or have been dropped. Data Type: Character, Read-only

Table 3–14. GLV Event Procedure REXX Variables

Variable Name	Contents
GLV.OLDVALUE	Contents: This is the value the global variable had before it was modified. Note: Standard REXX definitions apply to variables that have never been referenced before or have been dropped. Data Type: Character, Read-only
GLV.PROGRAM	Contents: This is the name of the program or event procedure that updates the global variable. Data Type: Character, Read-only
GLV.TEXT	Contents: This is the text message which describes the global variable update event. The string, truncated at 100 bytes, contains the `GLV.NAME`, `GLV.PROGRAM`, `GLV.OLDVALUE` and `GLV.NEWVALUE` values. Data Type: Character, Read-only
GLV.USER	Contents: This is an 8-byte field for communicating between event procedures that are executing for a single event. This field can be used to pass information between multiple event procedures. This field is initialized to binary zeroes. Data Type: Character, Read-write, maximum length 8.

Table 3–14. GLV Event Procedure REXX Variables

Refer to Chapter 8, “Automated State Management Facility (ASMF),” for more information.

Time-of-Day (TOD) Event Procedures

TOD event procedures are generated whenever the MVS timer associated with a TOD event procedure expires.

How Time-of-Day (TOD) Rules Work

The time or date you specified in a TOD rule's definition determines when the MVS timer expires.

TOD Event Procedure Criterion

The TOD event procedure criterion value is specified in the following form:

```
/*TOD todspec, interval, endspec, maxexecs
```

The following describe each of the components of the TOD event procedure criterion value.

Name	Description
todspec	**Time-of-Day Specifier.** Either the time-of-day specifier or the interval must be present to define a TOD event procedure. Follow these guidelines when specifying the todspec value: • Specify a date or time in any order. The format for specifying times and dates is given in Table 3–16 on page 3-19. • Use either upper or lowercase letters. • If the todspec value is omitted, code the comma following it to indicate its omission. An interval specification must be present if the todspec is omitted.
interval	**Execution Interval Specifier.** (optional unless TOD specifier omitted) The interval specifier give the amount of time units between event executions. If the interval specifier is omitted, the event procedure executes one time at the time value given by the time-of-day specifier. The valid formats for the interval specifier are given in Table 3–16 on page 3-19. If other specifications follow the interval, and the interval specifier value is omitted, code the comma following it to indicate its omission.
endspec	**Ending Time-of-Day Specifier.** (optional) The specifier gives the time/date value after which executions of this event procedure cease. The valid format for the endspec matches the todspec format.
maxexecs	**Maximum Executions Specifier.** (optional) The value for maxexecs is the number of times the event procedure is to be executed. This value is coded as an integer count value.

Table 3–15. TOD Criterion and Description

Specifier Formats

The following table gives the correct formats for coding each of the TOD event procedure specifier values:

Specifier Type	Format	Description
Date	Any one of these formats is acceptable: • dd MMM year • yy/mm/dd • weekday	The day, month, year, or day of the week, depending on the format used: • **dd:** A two-digit integer (01 through 31) corresponding to the day of the month • **MMM:** One of the following three-character abbreviations for the name of a month: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC • **year:** A four-digit year (for example, 1996) • **yy:** A two-digit integer corresponding to a year (for example, 96) • **mm:** A two-digit integer (01 through 31) corresponding to a month of the year. • **Weekday:** The full name of a weekday (for example, SUNDAY and MONDAY).
Time	hh:mm:ss	The time in 24-hour military format, as follows: • **hh:** A two-digit integer (00 through 23), indicating the hour • **mm:** A two-digit integer (00 through 59), indicating the minute • **ss:** (Optional) A two-digit integer (00 through 59), indicating the seconds after the minute.
Interval	**n units** Note: You can also use the 24-hour military time format described above.	The frequency and type of the interval: • **n:** An integer multiplier, indicating the number of interval time units • **units:** One of the following time units: DAY(S), WEEK(S), HOUR(S), MINUTE(S) or MIN(S), SECOND(S) or SEC(S) For example: 3 WEEKS 50 SECS 5 MINS 1 DAY

Table 3–16. Specifier Formats

TOD Event Procedure Header Keywords

No keywords are currently defined for TOD event procedures other than the specifier values previously stated.

TOD Allowed Process Sections

Only REXX-language process sections can be coded. Header-only rules are not allowed.

TOD Valid Return Values

The value returned from a TOD event procedure does not:

- Have any special meaning for the event facility.
- Have any effect on subsequent SEF processing.

TOD Event Procedure REXX Variables

When a TOD event occurs, the system extracts information about the event and creates event related variables. These variables are pre-instantiated when the TOD event procedure is scheduled for execution. You can access these data items directly, by name, from within REXX-language event procedures.

Variable Name	Contents
PHASE	Contents: This contains a four-byte character constant which indicates the processing phase for which the current event procedure was invoked. • If set to "INIT", the procedure is enabled either during subsystem start-up or in response to a user enable request. • If set to "PROC", the procedure runs after being matched to an actual global variable update event. • If set to "TERM", the procedure is disabled either during subsystem shut-down or in response to a user disable request. During procedure enablement and disablement, no other event relative variables are instantiated. Data Type: Character, Read-only
TOD.NEXTFIRE	Contents: This value indicates the next time the event procedure will run. Possible values are: • The date and time the rule fires next, in `yyyy/mm/dd hh:mm:ss` format • NONE, if the rule does not execute again Data Type: Character, Read-only
TOD.USER	Contents: This is an 8-byte field for communicating between event procedures which are executing for a single event. This field can be used to pass information between multiple event procedures. This field is initialized to binary zeroes. Data Type: Character, Read-write, maximum length 8.

Table 3–17. TOD Event Procedure REXX Variables

Type (TYP) Event Procedures

At the current time, TYP event procedures are reserved for use only by NEON Systems. They are used to develop prototypical Web server process section compiler/interpreter routines.

Warning:
You must not alter the distributed TYP event procedures in any way.

NEON Systems may elect to publish additional information about TYP event procedures at a later time. We strongly recommend that you do not code your own TYP event procedures, at this time

TYP Event Procedure Criterion

The criterion value must be a 1-to-8 byte process section "type" name. The name must be unique within the system.

TYP Event Procedure Header Keywords

No keywords are currently defined for TYP event procedures. Keywords cannot follow the criterion value on the event procedure header statement.

TYP Allowed Process Sections

Only REXX-language process sections may be coded. Header-only rules are not allowed.

TYP Valid Return Values

Unpublished, at this time.

TYP Event Procedure REXX Variables

Unpublished, at this time.

WWW Event Procedure Rules

This section expands on the concepts and information presented in *Shadow OS/390 Web Server's Getting Started Guide*, such as, If you do not find the information in this manual, please refer to the *Getting Started Guide* or one of the related publications in Shadow OS/390 Web Server's set of manuals.

For detailed information (such as, what is a WWW event, how to create one, input and output examples, etc.) refer to *Shadow OS/390 Web Server's Getting Started Guide*. This section expands on those concepts and information presented in it.

MVS, in its native form (catalog structure, file system naming conventions, JCL, EBCDIC code pages, and overall operating configuration), does not easily fit within the UNIX origins of the World Wide Web. Any Internet applications deployed on MVS must provide a means of translating UNIX-style requests into MVS resource designations, and then translating the responses into the expected UNIX-style formats and ASCII character sets.

Shadow OS/390 Web Server performs this UNIX-to-MVS resource mapping by using WWW event procedure rules.

How WWW Rules Work

WWW event procedure rules can:

- Define a native MVS dataset resource to be returned for the URL and provide the specifications needed for transforming the resource into an ASCII, browser compatible entity.
- Define the options needed to make use of a server provided turnkey interface for extracting and formatting data from legacy systems such as CICS, IMS, DB2, ADABAS, and TSO/E.
- Execute a script or customer written program which generates the response associated with the requested URL.

WWW Rule Types

There are three types of WWW rule definitions which can be defined within Shadow OS/390 Web Server.

- URL filter rules
- Target WWW rules
- Gateway filter rules

URL Filter Rules

A URL filter rule is a WWW rule which contains only the rule header statement (the "`/*WWW`" delimiter, URL match criterion, and optional header statement keywords, but not a process section). URL filter rules are normally defined generically using a wildcard as the last character of the URL match criterion.

How They Work

Header-only, URL filter rules are used to specify run-time transaction security limits or diagnostic attributes which control the MVS and server environment that are to be in effect during a subsequent procedure execution. The run-time attributes specified by URL filter rules are accumulated through each URL-to-rule match step. Each time a URL filter rule is matched, the server saves the updated attributes before it looks for another URL-to-rule match.

A header-only URL filter rule is not considered to be the ultimate target of any URL-to-rule search. URL filter rules can only set run-time security attributes and options; they do not cause an outbound response message to be generated.

Security

The server provides a Security Administration scheme which allows certain security related parameter values to be specified only within the master WWW ruleset. URL filter rules (within the master ruleset) can be used to control run-time security attributes of all the transactions in a subordinate WWW ruleset..

During rule lookup, the server must locate a target WWW rule definition, even if one or more URL filter rules were found. Otherwise, the URL requested by the client is considered to be undefined and a "URL Not Found" response is generated.

Whenever a procedural WWW rule is matched (the rule contains a process section), these accumulated run-time attributes are put into effect just before the procedure is executed.

Example

One URL filter rule attribute could specify whether the end user must provide a valid MVS userid and password before being allowed to request a URL. When the accumulated attributes are put into effect, the server determines if a valid logon has occurred. If it has, the procedure is executed; if not, the server generates a rescan to a recovery procedure.

Target WWW Rules

A WWW rule, which contains a process section declaration (following the "`/*WWW`" header statement line), is a normally a target WWW rule, unless the rule is explicitly declared to be a gateway filter rule.

How They Work

A target WWW rule's process section is executed whenever a URL-to-rule match selects a matching rule. Execution of the process section definition is expected to generate an outbound response to the client's original request.

Because target WWW rules generate all actual transaction responses, they are considered to be the ultimate targets of all URL-to-rule match searches. If no target WWW rule is matched while scanning for URL-to-rule matches, the URL is considered to be "`Not Found`".

Once a target WWW rule has been matched and executed, the server ceases to look for additional URL-to-rule matches, because the first target rule match is considered to have responded to the client request. Unless the target WWW rule requests rescan to a new URL value, the Web transaction ends following execution of the target rule.

Target rules execute under control of the security and environmental run-time options have been accumulated during previous matches to URL filter rule and gateway filter rules.

Gateway Filter Rules

A WWW rule definition which contains a procedure (such as, a process section), can be explicit declare to be a gateway filter rule. You make this explicit specification by coding the keyword, "GATEWAY", as an operand on the "`/*WWW`" header statement.

How They Work

Gateway filter rules are handled much like target WWW rules, except they:

- Are not assumed to generate a client response.
- Are not considered to be the ultimate target of a URL-to-rule match search. After a gateway filter rule is executed during URL-to-rule searching, the server continues searching until it locates a target WWW rule.
- Can re-direct subsequent processing using rescan or flush requests.

If a target WWW rule is not located by subsequent matching, the URL request is rejected with a "`Not Found`" condition even though the gateway procedure was executed. Gateway filter rules execute under control of the security and environmental run-time options that have been accumulated during previous matches to URL filter rules and gateway filter rules.

Gateway filter rules can be used to provide a common front-end procedure which is executed each time a generic URL request is made against a set of related Web transaction definitions. The gateway can monitor query variables or cookie values and use rescan or flush to re-direct processing until some validity requirement has been fulfilled. Once the validity check is passed, the gateway exits normally and allows the server to continue its search for a target rule definition.

Example:

A gateway filter rule can specify a REXX procedure or user written program that always executes before a match to a more specific target URL strings. For example, it can be used to implement a customized HTML form-based logon/ logoff front-end transaction which must be concluded before access to the target rules is allowed.

For such a front-end, the gateway rule is matched and executed first. It then checks to see if a valid logon has already been performed. If yes, the gateway filter rule terminates normally so subsequent processing by the server locates one of the target rules being front-ended. If the logon has not been completed, the gateway filter rule procedure could display and process a logon HTML form, which creates a timed token to remember the logon through subsequent interactions with the client.

Syntax of WWW Rule Definitions

Each WWW rule definition is coded within a separate member of a ruleset PDS(E) dataset. The syntax needed to define a valid WWW rule definition contains:

- A WWW rule header statement (required)

  This statement must begin in the first position of the first record within the rule (disregarding optional record numbering) and it must begin with the delimiter characters, "`/*WWW `".

  The URL match criterion is a required positional operand which must appear on the same line, immediately following "`/*WWW `". The match criterion string gives the URL value(s) which can be matched to this rule to trigger its selection during URL-to-rule searching. For example:

  ```
  /*WWW  /NEON/HTMLFILE/*  AUTHREQ(NO)
  ```

  Optional keyword parameters follow the URL criterion string on the first input line and can be continued on subsequent lines. These keywords define operational characteristics which are set up when a URL-to-rule match occurs. Keywords control the run-time security authorization framework and allow various diagnostic information to be selectively produced.

- A process section declaration (optional)

  WWW event procedures can contain:

  – Header-only rules (no process section).
  – REXX, FILE or PROGRAM process sections.

  Example:

  ```
  /*FILE DATATYPE(PDS) DDNAME(HTMFILE)
  CONTENTTYPE(text/html)-
  FORMAT(BINARY)
  ```

WWW Header Keywords

Many keyword parameters can be coded on the "`/*WWW `" rule header statement. All are optional, but if coded, they must follow the required URL matching criterion value.

The WWW header keywords fall into the following categories:

- Security Administration Controls
- Gateway
- Developer Selectable Options

Note:
If you continue a "`/*WWW` " header statement onto multiple lines, code a trailing dash (`-`) at the end of a line to indicate that the statement is continued onto the following line.

Security Administration Controls

The security keywords are used to configure the run-time authorizations under which a transaction operates. Some of these keywords can only be specified by a Web server administrator within the master ruleset and are not available to the developer unless delegated otherwise.

Security keywords are specifically related to server security administration and can be applied generically to groups of WWW rules. See Chapter 4, "Web Transaction Security," for more information on the following security administration control keywords:

- AUTHREQ(YES|NO|LOCK)
- RESOURCE (string)
- RUNAUTH(NONE|CLIENT|proxy-id)
- SSL(NO|COND|YES|LOCK|LOCKCOND)

GATEWAY

The GATEWAY keyword declares that the `/*WWW` rule act as a gateway filter rule. This GATEWAY keyword must be explicitly coded in order to designate all gateway filter rules.

- `/*WWW` rules that contain a process section, but do not have the GATEWAY keyword explicitly coded are always processed as target WWW rules.
- `/*WWW` rules that do not contain a process section are always processed as URL filter rules.
- The distributed rule in `'WWW.SWSCNTL'` is an application dependant example of a REXX language logon gateway filter rule which handles HTML form-based MVS logons. Use the URL `/SWSCNTL/MENU` to access this sample application.

Developer Selectable Options

Some "`/*WWW` " statement keywords specify developer selectable options can be used to control the transaction run-time environment and resource allocations needed for execution of an individual WWW rule. For instance, a developer may need to override the server's default limitation on maximum outbound response size if the rule is expected to output a very large response. Or a run-time tracing option may be temporarily turned on in order to trace an outbound response for debugging purposes.

The developer selectable options can be specified for any WWW rule and only apply to the an individual rule. The developer keywords are:

AUTHOFLUSH(integer)

This keyword parameter operates only for Web transactions that execute in "non-parsed header" mode; otherwise, it is ignored.

The server can automatically issue flush-to-client requests periodically while an outbound response is being composed. This parameter is the number of 32k outbound transmission buffers which must be filled before an automatic flush-to-client request is issued.

The operand value for this keyword must be in the range of 0 to 32767. Coding 0 (zero) disables the automatic flush-to-client feature. If this keyword is not explicitly coded for a rule, the start-up parameter MAXCHAINEDBUFFERS value is used instead.

DPRTY(value)

This keyword overrides the dispatching priority of the Web transaction subtask relative to other subtasks within the server address space and allows you to raise or lower the relative priority of certain Web transaction tasks. This is useful when defining long-running DB2-based transactions to keep those tasks from slowing overall response time.

Code the operand of DPRTY as a integer between 0 and 255 with or without a leading sign. Do not code blanks between the leading sign, if any, and the integer following it.

- If no leading sign is present, the operand value gives the absolute dispatching priority to be assigned to the transaction subtask.
- If a leading plus or minus sign is coded, the current dispatching priority is raised or lowered by the value given.

Only WWW procedures coded within the master ruleset may specify a net increase in a task's dispatching priority. Any attempt to increase the priority from within a subordinate ruleset is ignored.

HTXTRACE(list)

This keyword specifies whether a pre-compile or evaluation time trace is performed when an HTML extension processing occurs while executing a WWW rule. The trace options apply only to HTML extension processing that occurs while executing a WWW rule. The HTXTRACE option is not cumulatively applied to the WWW transaction when more than a single WWW rule is matched.

If this keyword parameter is not coded, the subsystem will not trace HTML extension processing while executing this or any other WWW rule.

This parameter can be used to diagnosis problems when processing source file information containing complex HTML extension statements. The keyword has no effect if HTML extensions are not processed while the transaction is executing.

Specify one to three of the following keyword values as the operand of the HTXTRACE parameter. Separate multiple keywords with a single space, not a comma. For example:

Parameter Value	Meaning
HTXTRACE(ALL)	Perform all traces. "ALL" is equivalent to specifying "`HTXTRACE(EMIT EVAL PCODE)`".
HTXTRACE(EMIT)	Trace internal P-code elements during compilation. If a cached p-code image is in use, the trace is not performed. In order to ensure that compile time p-code emit trace is performed, purge the cache storage for the member being processed before invoking the WWW transaction.
HTXTRACE(PCODE)	Trace internal P-code elements immediately before performing evaluation and substitution of the source file.
HTXTRACE(EVAL)	Trace evaluation of each HTML Extension statement as it is processed during output of the source file.

Table 3–18. HTXTRACE(EMIT EVAL PCODE)

MAXRESPBUFFERS(integer)

This keyword parameter specifies a limit on the number of 32k, outbound HTTP response buffers which any single transaction can simultaneously hold in storage while an outbound response is being composed. The minimum value that can be specified is 0, and the maximum is 32767. If 0 (zero) is coded, buffer-count limit checking is not performed.

If this parameter is not coded, the server uses the setting of the start-up parameter, MAXHTTPRESPBUFFERS, as a limit on the number of outbound response buffers. If this limit is exceeded, Shadow OS/390 Web Server generates a `User Abend x'722'` with a reason code 500 in order to cancel the transaction procedure.

MAXRESPBYTES(integer)

This keyword parameter specifies a limit on the total number of bytes that can be transmitted in an HTTP response message. The minimum value that can be coded is 0 (zero), and the maximum is 2147483647. If 0 (zero) is coded, outbound byte count limit checking is not performed.

If this parameter is not coded, the server uses the setting of the start-up parameter, MAXHTTPRESPBYTES, as a limit on the number of outbound bytes. If this limit is exceeded, Shadow OS/390 Web Server generates a `User Abend x'722'` with a reason code 500 in order to cancel the transaction procedure.

RESPMODE(SERVER | NONE)

This keyword parameter can be set to control the operational mode of the server while an outbound response is being composed by the Web transaction. Two operational modes are provided.

Parameter Value	Meaning
RESPMODE(NONE)	The Web transaction composes and transmits an outbound response in "non-parsed header" mode.
RESPMODE(SERVER)	The Web transaction composes an outbound response, and the server monitors and transmits the response in "server-parsed" mode.

Table 3–19. RESPMODE Parameters

If the RESPMODE keyword parameter is not explicitly coded, the server executes the Web transaction using the setting of the start-up parameter HTTPRESPMODE parameter. Server parsed mode operation is the preferred and it is the normal operational mode.

Non-Parsed Header Mode. Non-parsed header mode requires that the user supply all the required HTTP response headers needed to communicate with the client Web browser. In this mode, the server does not examine, override, or augment the outbound HTTP response message headers output by the Web transaction. The Web transaction procedure has direct control of the outbound communications session and can even have a partial response transmitted to the client before the entire outbound response message has been composed.

The server uses this operational mode when transmitting huge files to prevent large outbound responses from occupying excessive amounts of virtual storage. Refer to the *Shadow Programmer's Guide* or the online HTML for information on SWSSEND, high-level-language SWSSEND, ADDRESS SWSSEND and flush-to-client operations.

Following the completion of a transaction operating in non-parsed header mode, the server never attempts to provide persistent session support; the communications session is always closed.

Server Parsed Header Mode. In server parsed header mode, Shadow OS/390 Web Server buffers the complete outbound response until the Web transaction is completed. The server then examines whatever HTTP response headers that were explicitly output by the transaction. The generated transaction HTTP response headers could be left unchanged, altered, removed by the server, or additional HTTP response headers could be inserted into the outbound response message. After reviewing the HTTP response headers, the server transmits the complete outbound response message.

PARSETRACE(NO | YES)

This keyword specifies whether the parsing of “`WWW.`” event related variables is to be written to the wrap-around trace dataset (trace browse). If the parsed variables were not previously traced during execution of the Web transaction, the trace is written when the WWW rule is matched,

If this keyword parameter is not coded, the subsystem uses the setting of the product level parameter, TRACEURLPARSE, to determine if outbound tracing is performed.

The following values can be specified for PARSETRACE:

Parameter Value	Meaning
PARSETRACE(NO)	Do not generate a trace of the parsed `WWW.` variables.
PARSETRACE(YES)	Generate a trace of the parsed `WWW.` variables, if not previously traced.

Table 3–20. PARSETRACE Parameter

QUEUESIZE(integer)

This keyword specifies the number of entries allocated within the external data queue which is used by Shadow/REXX to support the **PUSH** and **PARSE PULL** instructions. The queue is also used to return the results of executing a TSO request within an external TSO server address space.

Use this keyword parameter to:

- Override the default external data queue specification made by the SEFMAXQUEUE start-up parameter.
- Increase the default queue allocation size when a large number of queued lines are expected in a response. If the value specified for the `QUEUESIZE( )` keyword is smaller than the SEFMAXQUEUE start-up parameter, the value set for SEFMAXQUEUE is used instead.

Code the operand of `QUEUESIZE( )` as an integer in the range of 1 to 1,000,000. The operand gives the absolute number of entries to be pre-allocated for the external data queue.

SENDTRACE(NO | YES)

This keyword specifies whether the transaction generates wrap-around trace entries to record outbound data sent to the Web client. If this keyword parameter is not coded, the subsystem uses the setting of the product level parameter, TRACEHTML, to determine if outbound tracing is performed.

The following values can be specified for the SENDTRACE:

Parameter Value	Meaning
SENDTRACE(NO)	Do not generate a trace of the outbound transmission.
SENDTRACE(YES)	Generate a trace of the outbound transmission.

Table 3–21. SENDTRACE Parameters

WORKSIZE(integer)

This keyword specifies the size (Kilobytes) of the REXX work space to allocate while executing this WWW rule. REXX work space is used by Shadow/REXX to contain run-time variable names, values, and evaluation areas during execution.

Use this keyword parameter to:

- Override the default work space size specification made by the SEFSIZE start-up parameter.
- Increase the default allocation size when a large number of run-time variables are expected. If the value specified for the `WORKSIZE( )` keyword is smaller than the SEFSIZE start-up parameter, the value set for SEFSIZE is used instead.

For example, you may find this useful when using the ADDRESS SQL environment of Shadow/REXXTOOLS. If a DB2 query returns a large result set, a REXX work space is needed to contain each column's variable name and value, which could result in large space requirements.

Code the operand of `WORKSIZE( )` as an integer in the range of 48 to 2,000,000. The operand value specifies the work space size to be allocated in kilobytes.

WWW URL-to-Rule Matching

The following discusses how the URL-to-rule matching and its criteria is handled.

Search Order

When the server begins a URL-to-rule search, event procedures are matched from the least-to-most specific. Unless a rescan is done, the search ends when the first target WWW rule is matched.

Note:
URL-to-rule matching can be prematurely suspended if any gateway or target procedure issues a flush request to suspend further rule matching and terminate the current transaction normally.

Match Order

If two or more WWW rules specify exactly the same URL match criterion string, they are matched in the following order:

- Rules defined in the master WWW ruleset are always matched before rules defined in subordinate WWW rulesets..
- URL filter rules, consisting of only a "`/*WWW `" header statement, are matched before any gateway filter rules or target WWW rules.
- Gateway filter rules are matched before any target WWW rules.

Warning:

It is possible to create a group of WWW rule definitions where some WWW rules can never be matched. Because each search proceeds from least-to-most specific match, and searching ends when a target is located, wildcards must be used carefully. The server does not warn of unmatchable WWW rule definitions.

For example: If two like target rules are defined, only one is matched-to and executed at run-time. The order in which the match occurs is unpredictable. URL criterion values must be constructed so one WWW rule does not "hide" other, more specific rules.

WWW Rule Header Statements

You can specify a 1 to 128 character string for the criterion value of the WWW event procedure rule header statement. The criterion must be coded as a continuous string of non-blank characters and must appear on the first line of the rule; continuation of the URL string is not allowed.

If the value is specified in lowercase, SEF converts it, internally, to uppercase. Shadow OS/390 Web Server processes all URL matching operations using uppercase URL values.

Character String Restrictions

- If the event procedure resides within the master WWW ruleset, the character string that you specify as the URL matching criterion is unrestricted.
- If the event procedure resides within the subordinate WWW ruleset, the character string that you specify as the URL matching criterion is restricted. The criterion value must begin with the string, "`/xxxxxxxx`", where "`xxxxxxxx`" is is the ruleset name. Any combination of characters is allowed after the required prefix.
- If a leading slash character is not part of the criterion specification (allowed only within the master ruleset), the URL value cannot be matched by an

externally entered Web transaction. Only a rescan operation can invoke an event procedure without using a leading slash character.

See Special Characters and URL String in the *Getting Started Guide* for recommendations on structuring URL criterion values.

WWW Rule Process Sections

Any of the valid process section types can be coded as part of a WWW rule to specify a procedure to be executed by the server.

- REXX sections for low-to-intermediate volume scripting of Web transactions
- FILE sections to return an MVS dataset as a Web transaction response
- PROGRAM sections for high-volume transaction deployment
- EXECSQL sections for rapid creation of Web transaction responses using data stored in a DB2 data base
- TSOSRV sections for executing a TSO/E command processor, CLIST, REXX, or ISPF procedure in an isolated out-board TSO/E server

WWW Rule Examples

Setting Run-Time Security Environment (URL Filter Rule)

The following example shows a generic filter rule set up a run-time operational limit for more specific URL-to-rule matches within the generic group.

```
/*WWW /NEON/* RUNAUTH(UserN)
```

A third-party-proxy userid, "`UserN`" is used to execute any URL-to-rule transaction procedure which begins with “`/NEON/...`”. The remaining examples (shown below) run within an MVS subtask where "`UserN`" is set as the run-time MVS userid.

Accessing Native MVS Files (Target WWW Rule)

Whenever a URL arrives in the system, it is matched to a WWW rule which is then executed by the server. For example:

In-bound URL:

```
HTTP://my.server/NEON/HTMFILE/MEMBER2
```

Matching WWW Rule

```
/*WWW /NEON/HTMFILE/*
/*FILE DATATYPE(PDS) DDNAME(HTMFILE) CONTENTTYPE(text/html) -
       FORMAT(TEXT)
```

In this example, the server's built-in `/*FILE` procedure executes to handle the client's request.

The /*FILE process section uses information represented by the URL wildcard character ("MEMBER2") to specify which PDS member to be returned to the client. Since this rule contains a procedural specification (the /*FILE process section), it is allowed as a target of the URL-to-rule match. A return response (to the client) is expected.

The prior URL-to-rule match for "/NEON/*" causes the proxy userid, UserN, to be set up while this procedural rule executes. See the previous example.

Turnkey DB2 Access Rule (Target WWW Rule)

This WWW rule is executed by the server whenever a matching URL request arrives.

```
/*WWW /NEON/QSTAFF AUTHREQ(NO)
/*EXECSQL SUBSYS(DSN2) -
PLAN(SWSCC1010) -
MAXBLOCKS(30) -
AUTOFORMAT( TITLE('Contents of Q.STAFF Table') -
BODY('bgcolor="#FFCC33"') -
)
select * from q.staff
```

Here, the server executes an SQL statement ("select * from q.staff") and then dynamically formats the result set as an HTML table. Additional process section parameters can be used to refine the SQL query or allow customized HTML output.

Authorization to access the Q.STAFF table is granted or denied based upon the run-time proxy userid, UserN, which was set up by the filter rule. See "Setting Run-Time Security Environment (URL Filter Rule)" on page 3-33.

WWW Event-Related Variables

Each time an inbound HTTP request is received by the Shadow OS/390 Web Server subsystem, the system parses the HTTP request header to determine which event procedure to run. During the parse operation, various data items are extracted from the transaction header along with other environmental data elements. These items are made available to WWW rule transaction procedures and is used in composing an HTML response.

Using the Shadow/Rexx Interpreter

By using the variable names listed in Table 3–22 on page 3-35, Shadow/REXX can access these run-time values by name. These variables are automatically set up in the REXX-language environment before a Shadow/REXX procedure is executed.

You can see a demonstration of how a Shadow/REXX procedure is invoked and can make use of these run-time variables by accessing the sample procedure /NEN/DEMO01.

Using Non-Shadow/REXX Interpreters

User written programs or REXX procedures executed by other (non-Shadow/REXX) interpreters must use an API interface call to retrieve the value of these run-time variables. Use the SWSVALUE API Call for user written high-level language program access, or the SWSVALUE built-in function for non-Shadow/REXX interpreters.

Variable Name	Contents
PHASE	Contents: Contains a four-byte character constant which indicates the processing phase for which the current event procedure was invoked. • If set to "INIT", the procedure is being enabled either during subsystem start-up or in response to a user enable request. • If set to "PROC", the procedure is being run after being matched to an inbound HTTP transaction request. • If set to "TERM", the procedure is being disabled either during subsystem shut-down, or in response to a user disable request. During procedure enablement and disablement, the only other variable that is instantiated is WWW.USER. The remaining variables are only instantiated during "PROC" phase processing. Note: REXX procedures normally run only during the PROC phase, unless explicitly requested. See Chapter 5, “Writing Web Transactions in REXX,” for more information. Data Type: Character, Read-only
WWW.ABENDCODE	Contents: The decimal value of the last encountered abend code. The value can be converted to displayable hexadecimal using the D2X built-in REXX function. The value is zero, if no abend has occurred during processing. Data Type: Integer, Read-only
WWW.ABENDREASON	Contents: The decimal value of the last encountered abend reason code. The value can be converted to displayable hexadecimal using the D2X built-in REXX function. The value is zero, if no abend has occurred during processing. Data Type: Integer, Read-only
WWW.ACCEPT_CHARSET	Contents: The value of the Accept-charset: HTTP request header token. If the Accept-charset: request header is not present in the inbound HTTP request, this variable is seto to a NULL value. Data Type: Character, Read-Only
WWW.ACCEPT_ENCODING	Contents: The value of the Accept-encoding: HTTP request header token. If the Accept-encoding: request header is not present in the inbound HTTP request, this variable is seto to a NULL value. Data Type: Character, Read-Only
WWW.ACCEPT_LANGUAGE	Contents: The value of the Accept-language: HTTP request header token. If the Accept-language: request header is not present in the inbound HTTP request, this variable is seto to a NULL value. Data Type: Character, Read-Only

Table 3–22. WWW Event-Related Variables

Variable Name	Contents
WWW.ACCEPT.0	Contents: Contains the number of Accept: HTTP request headers found within the inbound HTTP request. If none were found, the value of this variable is zero. Data Type: Integer, Read-Only
WWW.ACCEPT.n	Contents: Variables `WWW.ACCEPT.1` through `WWW.ACCEPT.n` (where n is equal to the value of the variable `WWW.ACCEPT.0`) contain each of the tokens set for the Accept: HTTP request headers found in the inbound HTTP transaction. If no Accept: headers were present in the inbound message, then `WWW.ACCEPT.0` will be set to zero, and the remaining `WWW.ACCEPT.n` variables will not be instantiated. Data Type: Character, Read-Only
WWW.AUTH	Contents: Set to a character value indicating the authorization level of the current event procedure. • If set to "NONE", no authorization data was sent with the inbound HTTP transaction request. The current transaction runs under the authorization of the Web Server subsystem or an overriding RUNAUTH userid value. • If set to "SENT", authorization data was sent with the inbound HTTP transaction request, but it was not used to perform userid/password validation. In order to conserve CPU resources, userid validation is only performed when required by the security attributes of the transaction. Because userid validation is not required by the current transaction, the authorization data sent by the Web client was not processed. The current transaction is running under the authorization of the Web server subsystem or an overriding RUNAUTH userid value. • If set to "NO", authorization data was sent with the inbound HTTP transaction request, but the userid or password value was invalid and could not be used to log on to the system. The current transaction is running under the authorization of the Web server subsystem or an overriding RUNAUTH userid value. • If set to "YES", the inbound HTTP transaction contained a valid userid and password, which were used to log on to the system. The current transaction runs under the authorization of the Web server subsystem, an overriding RUNAUTH userid value, or the authorization of the end user. Data Type: Character, Read-Only
WWW.AUTHDATA	Contents: Contains the value of the undecoded authentication information sent by inbound HTTP Authorization: request header. If no such request header was present, this variable contains a NULL string. Data Type: Character, Read-Only
WWW.AUTHMETHOD	Contents: Contains the value of the authorization method specified in the inbound HTTP Authorization: request header. If no such request header was present, this variable contains a NULL string. At present, the only allowable value for this variable is the string "Basic". Data Type: Character, Read-Only
WWW.AUTHMSG	Contents: Contains a string value containing the message issued by the security subsystem when the inbound userid and password were used to process a logon request. Data Type: Character, Read-Only

Table 3–22. WWW Event-Related Variables

Variable Name	Contents
WWW.AUTHORIZATION	Contents: Contains the decoded value of the Authorization: userid and password contained within the inbound HTTP request. If no Authorization: request header was present, this variable contains a NULL string. The password within the decoded authorization string has been overlaid with X's. Data Type: Character, Read-Only
WWW.AUXCOMPONENT	Contents: For some Web server processes, external MVS subsystems are invoked. If an exceptional condition occurs, the `WWW.AUXxxxx` variables are set to reflect the error. This variable contains the name of the external component. It is set to the string 'DB2' when exceptional conditions are noted during DB2 open processing or when processing an EXECSQL process section. Data Type: Character, Read-Only
WWW.AUXRC	Contents: For some Web server processes, external MVS subsystems are invoked. If an exceptional condition occurs, the `WWW.AUXxxxx` variables are set to reflect the error. This variable contains the return code set by the external component. Data Type: Integer, Read-Only
WWW.AUXREASON	Contents: For some Web server processes, external MVS subsystems are invoked. If an exceptional condition occurs, the `WWW.AUXxxxx` variables are set to reflect the error. This variable contains the reason code set by the external component. Data Type: Integer, Read-Only
WWW.AUXABEND	Contents: For some Web server processes, external MVS subsystems are invoked. If an exceptional condition occurs, the `WWW.AUXxxxx` variables are set to reflect the error. This variable contains the abend code set by the external component. Data Type: Integer, Read-Only
WWW.AUXOTHER	Contents: For some Web server processes, external MVS subsystems are invoked. If an exceptional condition occurs, the `WWW.AUXxxxx` variables are set to reflect the error. This variable contains any other code set by the external component. Data Type: Integer, Read-Only
WWW.AUXMSG	Contents: For some Web server processes, external MVS subsystems are invoked. If an exceptional condition occurs, the `WWW.AUXxxxx` variables are set to reflect the error. This variable contains text information describing the exceptional condition. Data Type: Character, Read-Only
WWW.COOKIE	Contents: Contains the value specified for the Cookie: HTTP request header, if present, or a NULL string. See the Netscape Documentation at Persistent Client State HTTP Cookies for more information. Data Type: Character, Read-Only

Table 3–22. WWW Event-Related Variables

Variable Name	Contents
WWW.COOKIE.xxxxx	Contents: Contains the value of the name/value pair, `xxxxx`, contained within the Cookie: HTTP request header. See the Netscape Documentation at Persistent Client State HTTP Cookies for more information. Data Type: Character, Read-Only
WWW.CONTENT_LENGTH	Contents: Contains the value specified for the Content-length: HTTP request header, if present, or a NULL string. Data Type: Integer, Read-Only
WWW.CONTENT_TYPE	Contents: Contains the value specified for the Content-type: HTTP request header, if present, or a NULL string. Data Type: Character, Read-Only
WWW.CURRENTURL	Contents: The current value of the URL being used to perform matching to Web event procedures. This is normally set to the value specified in the input HTTP transaction request, unless an intervening procedure or the subsystem has altered the match value. The subsystem alters match values when certain errors are encountered to re-direct processing to one of the built-in `SYSTEM/ERROR/nnn` procedures. User procedures can alter the match URL value by issuing "`RETURN RESCAN xxxx`" from an event procedure. Data Type: Character, Read-Only
WWW.DATE	Contents: Contains the value specified for the Date: HTTP request header, if present, or a NULL string. Data Type: Character, Read-Only
WWW.ERRORCODE	Contents: Contains the value of the transaction error code field. The field may be set by the subsystem in response to various transaction-related error events. Data Type: Integer, Read-Only
WWW.FIELD.0	Contents: Contains the number of field name/value pairs present within the input HTTP transaction. This value is set to zero, if no name/value pairs were present. Data Type: Integer, Read-Only
WWW.FIELD.n.NAME	Contents: Contains the name of the nth field name/value present within the input HTTP transaction. The value of n ranges from 1 to the value set for `WWW.FIELD.0`. Data Type: Character, Read-Only
WWW.FIELD.n.VALUE	Contents: Contains the value of the nth field name/value present within the input HTTP transaction. The value of n ranges from 1 to the value set for `WWW.FIELD.0`. If no value was present for the Nth pair, this variable contains a NULL string. Data Type: Character, Read-Only
WWW.FORWARDED	Contents: Contains the value specified for the Forwarded: HTTP request header, if present, or a NULL string. Data Type: Character, Read-Only

Table 3–22. WWW Event-Related Variables

Variable Name	Contents
WWW.FROM	Contents: Contains the value specified for the From: HTTP request header, if present, or a NULL string. Data Type: Character, Read-Only
WWW.IF_MODIFIED_SINCE	Contents: Contains the value specified for the If-modified-since: HTTP request header, if present, or a NULL string. Data Type: Character, Read-Only
WWW.INPUTURL	Contents: Contains the original inbound HTTP request URL value. Data Type: Character, Read-Only
WWW.LINE.0	Contents: Contains the number of individual lines within the inbound HTTP request header. Each line is delimited by a CRLF combination. Data Type: Integer, Read-Only
WWW.LINE.n	Contents: Contains the contents of the Nth line of the inbound HTTP request header. Each line is delimited by a CRLF combination. The line data does not contain the delimiting CRLF. The number of `WWW.LINE.n` variables is given by `WWW.LINE.0`. Data Type: Character, Read-Only
WWW.MATCHVALUE	Contents: Contains the criterion value of the current WWW event procedure to which the inbound URL was matched. Data Type: Character, Read-Only
WWW.MESSAGE_ID	**Contents:** Contains the value specified for the Message-id: HTTP request header, if present, or a NULL string. **Data Type:** Character, Read-Only
WWW.METHOD	Contents: Contains the value specified for the HTTP method specified in the inbound transaction. The Shadow OS/390 Web Server accepts transactions that specify the "GET", "POST", and "HEAD" methods. Data Type: Character, Read-Only
WWW.MIME_VERSION	Contents: Contains the value specified for the MIME-version: HTTP request header, if present, or a NULL string. Data Type: Character, Read-Only
WWW.PRAGMA	Contents: Contains the value specified for the Pragma: HTTP request header, if present, or a NULL string. Data Type: Character, Read-Only
WWW.PROTOCOL	Contents: Contains the value HTTP version value that was present within the inbound HTTP request header. Data Type: Character, Read-Only
WWW.QUERY	Contents: Contains the value of any encoded query data that was present within the inbound HTTP request header. Data Type: Character, Read-Only

Table 3–22. WWW Event-Related Variables

Variable Name	Contents
WWW.REFERER	Contents: Contains the value specified for the Referer: HTTP request header, if present, or a NULL string. Data Type: Character, Read-Only
WWW.SSL	Contents: Set to uppercase "Y" if an SSL connection is in use between the server and client. Set to uppercase "N", if an SSL connection is not in use. Data Type: Character, Read-Only
WWW.STATUSCODE	Contents: Contains the value set by a previous procedure of the subsytem for the HTTP response status code value. If set to zero, the subsystem substitutes value 200 (request was fulfilled). The only action of this variable is to save the status code placed into transaction-level SMF records. Data Type: Integer, Read-Write
WWW.TEXT	Contents: Contains the entire HTTP request header. Data Type: Character, Read-Only
WWW.USER_AGENT	Contents: Contains the value specified for the User-agent: HTTP request header, if present, or a NULL string. Data Type: Character, Read-Only
WWW.USERID	Contents: Contains the userid, if any, present within the inbound Authorization: request header. Data Type: Character, Read-Only
WWW.VAR.xxxxx	Contents: Contains the value of the input variable for each inbound query variable named xxxxx. If multiple query variables of the same name are input, the variable contains the value set only for the last of these. Data Type: Character, Read-Only

Table 3–22. WWW Event-Related Variables

CHAPTER 4: Web Transaction Security

This chapter expands on the concepts and information presented in *Shadow OS/390 Web Server's Getting Started Guide*, which includes, things you should know about Web browsers, the controlled transaction paradigm, levels of security, types of transactions, distributed transaction administration (master and subordinate rulesets, what they are and how they work), security attributes processing, security processing steps, implementing distributed transaction administration, and specifying web transaction security parameters.

About Web Browsers

Shadow OS/390 Web Server allows users with Web browsers to:

- Access DB2 tables via dynamic or static SQL using NEON's high performance, built-in DB2 connection facilities.
- Use CICS, IMS, and TSO/E transaction processing facilities.
- Access Web transactions that use DB2, VSAM, or virtually any data storage facility of MVS. Access is performed under strict controls imposed by the Server using services such as RACF, ACF/2, and TopSecret.
- Use customizable Web transaction services built-in to the Web server's rule-based architecture. The built-in applications include facilities for creating Web enabled applications that:
  - Serve static or run-time generated HTML or other data from virtually any MVS file. This function implements the intrinsic file server model used for HTTP, but allows mapping of URL requests to the native MVS file system.
  - Handle form based DB2 queries and updates.
  - Execute commands and procedures that execute within a TSO/E server, managed by the product. This facility provides access to various MVS data source and APIs that are normally unavailable using any other MVS based Web server. You can create Web enabled access to facilities such as ISPF, DFHSM, CLISTs or any other component which operates under TSO/E.
- Execute customer written transaction programs or scripts. You can write transaction programs using COBOL, PL/I, C, C++, Assembler or execute command procedures written in IBM TSO/E REXX language or a third-party vendor scripting language, such as Prevail/XP OPS/REXX.

Userid Prompting

The following information applies to the use of the “BASIC” authentication protocol supported by all Web browsers, but might not apply to newer authentication protocols.

Part or all of an outbound response to a URL request is an HTTP Response Header, which contains a standard status code value.

Response Status Code 401

When the client receives a response status code 401, it means the URL request is not authorized. Most browsers display a popup window that requests a userid and password. Once the user enters this information, the browser retransmits the original request, but adds an authentication header containing the encoded information.

The browser retains a copy of this information and automatically sends the same userid/password combination in response to subsequent 401 status errors. The browser continues to use this information until the browser session is terminated.

Re-logons

Because the browser retains the userid/password information, it makes a generic “re-logon”, such as you might use to switch TSO/E userids using a 3270 session, virtually impossible to provide. Generic re-logons require that the browser be closed and reopened. This appears awkward to users unless it can be customized to specific transactions.

At present, there is no way in which a server can signal a Web browser to drop its internal copy of a userid and password.

Each unique (and valid) userid, password, server domain, URL, and REALM combinations:

- Is cached by the browser. By using this cached information, the browser avoids re-prompting the user for a userid and password.
- Is considered valid by a browser if the server responds to a request with a “normal” status code (codes in the 2xx range).

Varying the “REALM” Value

By varying the “REALM” value transmitted in the HTTP 401 (`Not Authorized`) response, the server can force the end user to re-enter the information. However, the combination of userid, password, server domain, URL and REALM is not the original, and a new cache entry is generated.

This is not a useful means of providing generic solutions to the conflicts unless you want the end user re-prompted for every request. Except for very specialized transaction processes, the “REALM” value sent in 401 responses should be constant in relationship to the URL string.

Example

The following scenario cannot be resolved by the server unless the browser is closed after the password update is performed.

1. The client selects a given URL. For example, `/ABC`.
2. Because the `/ABC` rule requires authorization, the server responds with a 401 status code (`Not Authorized`).
3. Since this is the first request, no userid/password is cached. The browser presents the popup window and solicits a userid and password. The browser resubmits the original request for `/ABC`, but this time it includes the userid/password values.
4. The server processes the userid/password, determines they are valid, and allows the `/ABC` URL to be executed.
5. With the Web browser remaining open between requests, the end user re-selects the URL, `/ABC`.
6. Because the userid/password was previously valid for access to `/ABC`, the browser includes the original userid/password with the request.
7. When the transaction arrives at the server, the userid/password are again authenticated. However, because of the time delay, the password has expired.
8. The server transfers control to the password expiration URLs supplied with the product.
9. The password expiration URLs are executed successfully and the MVS password is updated.
10. At this point, if the browser is not closed and restarted, problems occur.

The Problem

1. The `/ABC` URL is re-selected by the end user.
2. The browser transmits the original userid and password, not the updated password.
3. Because the password is invalid, the server responds with a 401 (`Not Authorized`) code.
4. The browser retransmits the `/ABC` request, again using the old password, which it does without notifying the end user or displaying a message.
5. For older browsers, a loop starts in which the server constantly issues a 401 response and the browser constantly retransmits the same information.

Controlled Transaction Paradigm

The “controlled transaction” paradigm means Shadow OS/390 Web Server only performs those transactions in which you explicitly define what action takes place in response to an inbound URL based request. This includes which data files or programs are available to Web clients and under what conditions and authorization requirements each transaction is processed.

All gateways into your MVS system are closed unless you explicitly open them. This allows you to protect your system from unauthorized use by controlling which MVS resources are made available to the Internet.

Note:
Shadow OS/390 Web Server does not automatically return data files or execute programs based upon a user's request to obtain them; there is no intrinsic link between inbound URL values and the MVS file system. If a URL match value has not been specified within a WWW event procedure, the URL is rejected with an Unknown URL error message.

Levels of Security

There are several levels of security.

MVS Security Subsystem

Shadow OS/390 Web Server runs each Web transaction under the authorization of an MVS userid. When any Web transaction references MVS resources, Shadow OS/390 Web Server uses your existing MVS security subsystem to authorize access to the resource. The term userid is used within this document to refer to a valid MVS userid.

Client Authorization (Optional)

The server contains facilities that require authentication information to be provided with each inbound URL based request, and for validating the userid and password supplied by the Web client.

Authentication of client userids is optional, and can be selectively turned on or off for each transaction or transaction group by using the AUTHREQ parameter in the `/*WWW` header statement.

Conditions under Which a Client Userid is Required

Once the accumulated security attributes have been merged, a valid client userid and password are required whenever:

- `AUTHREQ(YES)` or `AUTHREQ(LOCK)` is in effect.

- `RUNAUTH(CLIENT)` is explicitly specified (even if `AUTHREQ(NO)` is also in effect).
- At least one matching rule has specified a value for the RESOURCE parameter. `AUTHREQ(YES)` is assumed in this case, since the client userid must be present and valid in order for the generalized resource check to be performed.

To conserve CPU resources, if the client userid is not required for authorization of a Web transaction, no authentication is performed. This is true even if the Web browser transmits an inbound authentication request header.

How an Effective Userid is Determined

A Web transaction's effective run-time userid is set based on the following logic:

- If the RUNAUTH keyword is explicitly specified for any matched WWW rule, the last valid value for RUNAUTH is used to determine the transaction's run-time effective userid.
- In the absence of an explicitly specified RUNAUTH keyword, the transaction runs under the authorization of the client userid, whenever the client userid is required.
- In the absence of an explicitly specified RUNAUTH keyword, and when the client userid is not required by the transaction authorization procedure, the default userid is used as the effective run-time userid.

Selective Access to URLs

By using a generalized resource rule protection scheme, access to URLs can be limited so underlying transaction procedures can only be executed by a subset of your MVS users. This allows you to restrict URL access by department, group or some other grouping.

These generalized resource rules either grant or deny access to individual URLs, usually by exploiting authorization controls already known to the MVS security subsystem, such as RACF and ACF/2.

The client userid is validated against the security subsystem resource name, which is specified by the RESOURCE parameter in the `/*WWW` header statement.

Effective Userid

Before any Web transaction procedure is executed, Shadow OS/390 Web Server ensures the Web transaction subtask is paired with an MVS userid (the effective userid). Using these subtask security controls, MVS determines what the Web procedure can do and which system resources it can and cannot access.

Shadow OS/390 Web Server has three possible sources for the run time effective userid. They are:

The Web Transaction Default Userid

This is the userid associated globally with Shadow OS/390 Web Server subsystem's address space; it is used for authorization of public Web server transactions. The default userid is normally set up with extremely limited authorization.

The default userid is specified by the parameter WWWDEFAULTRUNAUTH during product startup. If no startup option is given, the server uses the userid associated with the product started task address space.

Note:
We strongly recommend that you set up a special default userid for Web server transaction processing rather than using the server's started task userid.

Web transactions run under the authorization of the default userid when no other effective userid specifications have been made for a given URL value. (Most of the sample URLs supplied with Shadow OS/390 Web Server operate under the authorization of the default userid because they provide public information and demonstrations.)

The Client Userid

Based on the WWW rule options set, Shadow OS/390 Web Server can require a valid MVS userid/password combination from users. This means Web transaction processes can operate under the authorization and control of the end user's MVS userid.

Note:
Under some conditions, the client userid is required, even when the Web transaction procedure operates under control of the default or proxy userid. In cases like this, the client userid is only used to verify the user's permission to execute the transaction and not to control resource access while the transaction executes.

A Third-Party-Proxy (or 'RUNAUTH') Userid.

A third-party-proxy userid allows you to specify the run-time effective userid under which the Web transaction is processed. This is done without granting access to those resources to the default userid (for public access), other individual MVS userids or groups (for client userids). In most cases, the user is not even need to be aware of its use.

Warning:

Shadow OS/390 Web Server only allows third-party proxy userids to be specified by the RUNAUTH keyword within the master ruleset. Neither a password nor an authentication procedure is needed to log the userid on to the system. You must maintain close control over the master ruleset to guard against its misuse.

Shadow OS/390 Web Server assumes the default userid possesses a very low authorization level. Never create the default userid with a higher authorization level than intended. If you do, the overall administration of the system is compromised. (`RUNAUTH(NONE)` can be specified from both the master and subordinate rulesets.)

Security Option Summary

To define transaction level security processing (and therefore which MVS resources can be accessed) use a combination of parameters coded on WWW event procedure header statements. You can:

- Require the Web browser to provide a valid MVS userid and password as part of the HTTP transaction request header. Use the AUTHREQ parameter to specify this value.
- Require that a valid MVS userid, which is authorized to access a URL or group of URLs, be entered at the Web browser. To do this, check the client’s access privileges against a security subsystem generalized rule value representing the URL request. The generalized rule value is specified by the RESOURCE parameter.
- Specify the run-time effective userid under whose authority each Web transaction runs. If you do not explicitly specify the id using the RUNAUTH parameter, Shadow OS/390 Web Server defines it.

Distributed Transaction Administration

Deployment of Web server transaction definitions, particularly when made available to the World Wide Web, normally require security related issues to be tightly controlled. Larger installations might find it difficult to ensure proper security, especially with different departments and people responsible for deploying Web transaction definitions.

Shadow OS/390 Web Server’s Distributed Transaction Administration was designed to prevent accidental or malicious misuse of security related parameters, while still allowing diverse groups to have responsibility for writing and maintaining Web transaction definitions. Plus, it aids in administering both distributed and centralized transaction groupings.

The Master Ruleset

Web transaction definitions can be stored in one or more PDS datasets. Shadow OS/390 Web Server implements the Distributed Transaction Administration model by designating one of these datasets as the “master ruleset”. This designation (the mid-level qualifier name) is obtained from the start-up parameter, WWWEPROSET. If a value is not set for the parameter, the default is “WWW”.

The master ruleset contains:

- Procedures that were distributed with Shadow OS/390 Web Server and are required for proper operation.
- A definition for the home page URL value (the URL containing only a single slash (“`/`”) character).

How It Works

The master ruleset is designed to have limited accessibility and centralized administration because it is intended to control the security attributes assigned to transaction definitions that reside in subordinate rulesets. For this reason limit access to trusted personnel who are responsible for security administration of the Web server. Occasionally, you may need to grant limited access to a Web transaction programmer tailoring NEON supplied definitions.

In a highly centralized environment, where tight security and administrative control can be maintained, all Web transaction definitions can be placed in a single dataset. However, a master ruleset must still be designated. The subsystem aborts during start-up if 1) a master ruleset is not designated, or 2) the designated master ruleset cannot be opened.

Coding Master WWW Rulesets

The following rules apply to Web transaction definitions that reside within the master ruleset:

- The URL matching criterion can be composed of any 1 to 128-byte character string. There are no restrictions upon the criterion value.
- All transactions defined within the master ruleset run with the following security options in effect (unless explicitly overridden by parameters on the `/*WWW` header statement):
  - No generalized resource rule value is set. This means the URL value does not undergo a generalized resource rule check.
  - If the subsystem startup parameter, WWWDEFAULTAUTHREQ, is set to NO (where `AUTHREQ(NO)` is the assumed default value), then:
    - Client authentication information is not required, nor is it processed if received. It has no affect upon permission to execute the URL. (The URL is fully public).

    - The effective userid for the transaction is the Web server's default userid.

- If the subsystem startup parameter, WWWDEFAULTAUTHREQ, is set to YES (where `AUTHREQ(YES)` is the assumed default value), then:
    - Client authentication information is required and it must be evaluated as a valid MVS userid/password. The URL can only be executed if the end user has a valid MVS userid. If not, the transaction is rejected with a "`Not Authorized`" error.
    - The client userid is used as the Web transaction's run-time effective userid.
- Some security-related parameter values are only valid when coded within the master ruleset and cannot be specified within a subordinate ruleset. Specifically:
    - A third-party proxy userid can only be specified for the RUNAUTH parameter by a rule within the master ruleset.
    - AUTHREQ is always honored within the master ruleset regardless of its operand value. (`AUTHREQ(NO)` is ignored under some conditions when specified in a subordinate ruleset).
- At subsystem startup time, transactions defined within the master ruleset are enabled before transactions that reside in any subordinate ruleset. (This prevents timing-related security exposures during subsystems startup.)

Subordinate Rulesets

You can use a master ruleset to define all your Web transactions. However, if other Web definition datasets exist, they are automatically designated as subordinate rulesets. These rulesets (and the Web transactions defined within them) are subordinate to the security option controls specified in the master ruleset.

How They Work

Subordinate rulesets are intended for use on a departmental basis, where each department has its own subordinate rulesets which it controls. Security attributes are setup by centralized administration to ensure the transactions defined within a subordinate ruleset cannot accidentally or intentionally misuse the subsystem's special capabilities as an APF-authorized started task.

Coding Subordinate WWW Rulesets

Web transaction definitions, which reside within a subordinate ruleset, are subject to the following restrictions:

- The URL matching criterion specified for each transaction definition must begin with the characters "`/xxxxxxxx`", where "`xxxxxxxx`" is the ruleset name. This restriction directly relates each defined URL value back to the ruleset in which it resides.

Note:
This restriction:

- Eliminates the potential for duplicate URL values defined, or impersonation of URLs between datasets.
- Ensures that each departmental unit has complete control over its own portion of the URL name space apart from the security administration function performed by personnel authorized to update the master ruleset.
- Provides a crucial linkage between individual URL values and the MVS dataset name. This linkage ensures that only those end users which have read/write access to the subordinate rule datasets can define/alter Web transactions with certain URL values.

- Only a subset of the possible security related parameter option values can be specified for transaction definitions that reside within a subordinate ruleset. This limits some security processing attributes to the administrators of the master ruleset. Specifically:
  - A third-party proxy userid can not be specified as the operand of the RUNAUTH parameter, since this parameter can only be used in the master ruleset.
  - `AUTHREQ(NO)` is ignored during processing of the Web transaction, if a higher level, generic rule in the master ruleset was specified, like `AUTHREQ(LOCK)`.

Security Attributes Processing

Attributes Accumulated During the URL to Transaction Search

When a new URL arrives in the system, or when a rescan event occurs while processing a Web transaction, the value of the current URL is matched to all enabled WWW event procedures.

Match	Results
None found	The system generates a rescan event to a URL (SYSTEM/ERROR/404) which transmits a "URL Not Found" status.
To generic header-only rule	The security attributes specified by the latest match are merged to the overall attributes of the transaction. The search continues for additional matching rules.
To generic rule only (no other matches)	The system generates a rescan event to a URL (SYSTEM/ERROR/404) which transmits a "URL Not Found" status. This happens because no procedural specification was found to process the URL request.
First match to non-generic rule	The security attributes, if any, of the latest match are merged to the accumulated attributes. The non-generic rule is then processed by applying the attributes, performing security checks, and, if authorized, executing the defined procedure.
Searches for additional matches	These always cease once a non-generic rule is located. However, if a rescan event is generated by the transaction itself, or by the subsystem as part of an error recovery procedure, the matching procedure is re-inaugurated with the new current URL value.

Table 4–1. Possible Matches for URL to Transaction Search

Security Attribute Example

Refer to the event procedure libraries distributed with Shadow OS/390 Web Server while you read this section. The product distribution master ruleset library contains the following member, 'SWSCNTL'.

```
/*WWW /SWSCNTL* AUTHREQ(LOCK) RUNAUTH(CLIENT) RESOURCE(SWSCNTL)
```

When a URL for the product parameter displays/alters a transaction, /SWSCNTL/PARMS arrives. It is matched to the master ruleset procedure before it is matched to the more specific rule defined in the 'SWSCNTL' subordinate ruleset in the member 'PARMS':

```
/*WWW /SWSCNTL/PARMS
/*REXX
...remainder of REXX-language procedure
```

The security attributes specified by the first match (made to the generic value, /SWSCNTL*, in the master ruleset) are merged to the transaction specification at the time the first match is made. Actual processing of specified attributes is not performed at this stage.

The transaction level attributes are carried forward to point at which the second match is made. (The second match is made to the specific value, /SWSCNTL/PARMS, within the subordinate 'SWSCNTL' ruleset) is made.

Note:
Because the second match is to a rule which contains a procedural specification, authorization to execute the procedure along with the internal operations required to set up the run-time effective userid relationship must be performed. At this point (and not during the previous attribute merge operation), the accumulated security attributes are applied to the transaction.

Application of Security Related Attributes

During the URL matching search, if multiple generic header-only rules are defined, various transaction level security attributes can be toggled on and off. However, once matching activity crosses the boundary between master ruleset and subordinate ruleset, only a subset of security related values can be altered, regardless of the specifications made later.

- **Non-overrideable.** These security related attributes, which were set by generic rules defined within the master ruleset, remain in effect and govern the execution of a procedure matched within a subordinate ruleset.
- **Overrideable.** These attributes are set to the last specification made for the attribute.

Application of security attributes is deferred until late in the processing of each Web transaction to ensure that MVS security product, such as RACF or ACF/2, processing overhead is not incurred unless it is actually required.

Security Processing Steps

When processing each inbound URL request, Shadow OS/390 Web Server performs the a series of security related steps in the following order:

1. If authentication information was supplied with the inbound HTTP request, the userid and password values are parsed out, but not verified at this stage. (It is not known at this stage if authentication is required to execute the transaction.) If the authentication is not required, Shadow OS/390 Web Server completely bypasses processing this information to conserve CPU resources.

2. The security processing attributes of the transaction are set to known default values, which are:

   a. The AUTHREQ attribute of the transaction is set to the value specified by the subsystem start-up parameter WWWDEFAULTAUTHREQ.

   b. The RUNAUTH attribute is reset to indicate that no explicit RUNAUTH specification was made. In the absence of an explicit RUNAUTH specification, the run-time effective userid is determined. See “How an Effective Userid is Determined” on page 4-5.

   c. The Generalized Resource Entity value is set to blanks to indicate that no resource authorization check is performed.

3. The URL value is matched to one or more WWW event procedure definitions.

   a. If no match for the inbound URL was defined, the transaction is rejected by rescanning to the supplied “`SYSTEM/ERROR/404`” URL to transmit an “`Unknown URL`” error message.

   b. If a header-only WWW rule is matched, any security processing parameters specified by the header-only rule are merged to the transaction's security processing attributes. The security options are not applied at this time.

4. Once a WWW rule, which contains a procedural specification (a definition for an action to take place) is matched, Shadow OS/390 Web Server applies the security attributes to determine:

   a. If a client userid/password is required to access and execute the selected transaction procedure.

   b. If the generalized resource entity name was set to a non-blank value. If it was set to a non-blank value, the client must supply a userid/password, even if it was not required by other security related attributes.

   c. Which userid was set up as the run-time effective userid for processing the transaction.

5. Once these determinations are made, the server performs a logon of the client userid, if required.

   – If the client userid was not supplied or is invalid, the transaction is rejected by rescanning to the supplied “`SYSTEM/ERROR/401`” URL which transmits an “`Unauthorized`” error message. (If the client password has expired, it rescans to the “`PASSWORDEXPIRED`” URL.)

6. If the client userid was required, it is validated against a generalized resource rule to determine if the user is authorized to access the URL.

   – If the Client is not authorized, the transaction is rejected by rescanning to the supplied “`SYSTEM/ERROR/403`” URL which transmits a “`Forbidden`” error message.

7. The effective userid value is set up in system control blocks. This allows the transaction to run under the requested authorization.

8. The transaction's procedure definition is executed under the control of the effective userid.

9. If the transaction definition or subsystem detected error conditions cause a “rescan” operation to occur, the entire procedure is restarted at step 2.

See the *Shadow OS/390 Web Server User’s Guide* “Recovering From Server Detected Errors”. It explains recovery actions taken by the server for specific error conditions.

How to Implement Distributed Administration

The general method for implementing Distributed Administration is to:

1. Determine what default AUTHREQ/RUNAUTH value to start each URL matching procedure with, then set the corresponding value for the subsystem start-up parameter WWWDEFAULTAUTHREQ. For example:
   a. If WWWDEFAULTAUTHREQ is set to 'NO', then `AUTHREQ(NO)` is the default security attribute.
   b. If WWWDEFAULTAUTHREQ is set to 'YES', then `AUTHREQ(YES)` is the default security attribute.
2. Create an MVS Userid to be used as Shadow OS/390 Web Server’s Default Userid, then specify this Userid as the value for the start-up parameter WWWDEFAULTRUNAUTH.

   Note:
   The Default Userid should not have write access to the Shadow OS/390 Web Server’s LOAD library or to any of the server's SEF event procedure datasets.

3. Restrict access to the Shadow OS/390 Web Server’s master ruleset. Administrative personnel require **UPDATE** authority to the dataset but all other users should be restricted to **READ** access or prohibited entirely from accessing the dataset.
4. Place a generic, header-only WWW rule within the master ruleset that governs each of the subordinate WWW rulesets.
5. Enable each generic definition. Insure that the auto-enable event procedure attribute is set, that way the rule is re-activated each time Shadow OS/390 Web Server is restarted.
6. Code security options on each generic rule to set up security attributes to govern all WWW procedures residing within the specific subordinate ruleset being controlled. Non-generic URL transaction definitions in each subordinate ruleset can only specify whatever security overrides are allowed by the generic rule in the master ruleset.

Specifying Web Transaction Security Parameters

Web Transaction authorization capabilities are described in the Security Overview. This section gives an overview of how to specify Web Transaction security parameters. Refer to the *Shadow OS/390 Web Server User's Guide* for more information on these parameters.

Security parameter keywords are used to specify security authorization and attributes of Web transactions. They can be specified on the event procedure header statement for any WWW rule.

WWW Header Statement Keywords

A security-related parameter can be coded on any WWW event procedure header statement. Each parameter is optional.

The following example illustrates how security parameters can be coded:

```
/*WWW /NEON/INLINE   AUTHREQ(YES) RUNAUTH(CLIENT)
/*REXX
....a REXX-language procedure
```

WWW Header Keywords

Many keyword parameters can be coded on the "`/*WWW`" rule header statement. These keywords are optional, but if coded, they must follow the required URL matching criterion value.

Note:
If you continue a "`/*WWW`" header statement onto multiple lines, code a trailing dash (`-`) at the end of a line to indicate that the statement is continued on the following line.

Security Administration Controls

Other "`/*WWW`" statement keywords specify security administration controls that are used to configure the run-time authorizations under which a transaction operates. Some of these security related keywords can only be set by a Web server administrator and are not available to the developer unless access has been given to the master WWW ruleset.

These keywords are specifically related to server security administration, but can also be applied generically to groups of WWW rules.

Parameters	Description
AUTHREQ(YES|NO|LOCK)	Defines whether authentication of the client userid and password is explicitly required for transaction execution. Actual client userid authentication can be implied through the action of other security related options.
RUNAUTH(NONE|CLIENT|proxy-id)	Defines explicitly the run-time effective userid under which a Web transaction procedure runs.
RESOURCE (string)	Defines the generalized resource used for authorization to run the specific URL.
SSL(NO|COND|YES|LOCK|LOCKCOND)	Rejects, conditionally or unconditionally, and attempts to execute a Web transaction procedure unless a Secure Sockets Layer (SSL) session is in use between the server and client (that is, encryption of the communications session is a requirement). See the *Shadow Installation Guide* for information on configuring a Secure Socket Layer.

Table 4–2. Security Administration Control Parameters

See “Event Procedure Header Keywords” in the *Shadow OS/390 Web Server User’s Guide* for more information on security and non-security related WWW rule keywords.

Configuring Secure Sockets Layer (SSL) Support

See the *Installation Guide*.

WWW Header Security Parameters and Keywords

The following are security parameters:

- AUTHREQ
- RUNAUTH
- RESOURCE
- SSL

AUTHREQ (YES | NO | LOCK)

AUTHREQ parameter and keyword specify whether end user authentication is required to run the event procedure. If authentication is required, the end user must use the browser to provide a valid MVS userid and password before the server executes the defined transaction.

The following values can be specified for AUTHREQ:

Parameter Value	Meaning
AUTHREQ(NO)	The can be specified in either the master or a subordinate ruleset. The client userid authentication processing is not required. If authentication information was transmitted with the inbound URL, it is not processed. Under certain conditions, client userid authentication can still be required, even when `AUTHREQ(NO)` is in effect. When `AUTHREQ(NO)` is specified, and no explicit RUNAUTH specification is made, `RUNAUTH(NONE)` is implied. `AUTHREQ(NO)` is always honored when specified in a WWW master ruleset, but it is ignored if it is used in a subordinate ruleset and a previous URL match has set `AUTHREQ(LOCK)` into effect.
AUTHREQ(YES)	This can be specified in either the master or a subordinate ruleset. The client userid and password value must be supplied with the inbound transaction request, which is validated by the MVS security subsystem. In the absence of an explicit RUNAUTH specification, `RUNAUTH(CLIENT)` is implied by `AUTHREQ(YES)`.
AUTHREQ(LOCK)	This can only be specified within the master ruleset. If specified within a subordinate ruleset, the WWW header statement is flagged with an error and the transaction definition is not enabled. `AUTHREQ(LOCK)` is similar to `AUTHREQ(YES)`, except it cannot be overridden by any rule within a subordinate ruleset. If `AUTHREQ(LOCK)` has been set as a transaction level security attribute, AUTHREQ specifications made by subordinate ruleset rules are ignored and `AUTHREQ(LOCK)` remains in effect. `AUTHREQ(LOCK)` can be overridden by different AUTHREQ setting, only if it is changed by matching to a WWW master ruleset rule.

Table 4–3. AUTHREQ Parameters and Values

RUNAUTH(NONE | CLIENT | proxy-id)

The RUNAUTH parameter and keyword specify which run-time effective userid is used to execute the Web transaction. Default values for RUNAUTH are implied by the AUTHREQ parameter. You can override these defaults by explicitly coding the RUNAUTH keyword.

The following values can be specified for RUNAUTH:

Parameter Value	Meaning
RUNAUTH(NONE)	This can be specified in either the master or a subordinate ruleset. It overrides any previously set or implied RUNAUTH attribute. `RUNAUTH(NONE)` indicates the transaction procedure executes using the default userid as the effective userid. The default userid is specified by the startup parameter `WWWDEFAULTRUNAUTH`. If no value is specified, the userid associated with the Shadow OS/390 Web Server address space is used. If no matching transaction definition specifies RUNAUTH and the transaction attributes are set to `AUTHREQ(NO)`, then `RUNAUTH(NONE)` is the implied.
RUNAUTH(CLIENT)	This can be specified in either the master or a subordinate ruleset. It overrides any previously set or implied RUNAUTH attribute. `RUNAUTH(CLIENT)` means that the client userid and password value, which is validated by the MVS security subsystem, must be supplied with the inbound transaction request. The Web transaction is executed with the client's MVS userid set as the effective userid. `AUTHREQ(YES)` always implies `RUNAUTH(CLIENT)` unless another explicit RUNAUTH specification is in effect. Code `AUTHREQ(YES)` and omit `RUNAUTH(CLIENT)` unless you need to override a previously matched-to explicit `RUNAUTH(NONE)` or `RUNAUTH(xxxxxxxx)` specification.
RUNAUTH(xxxxxxx)	This can only be specified in the master ruleset. If `RUNAUTH(xxxxxxx)` is used within a subordinate ruleset, the WWW header statement is flagged with an error and the event procedure is not enabled. `RUNAUTH(xxxxxxx)` specifies the MVS userid to be used as a third-party proxy. The Web transaction is executed using this userid as the effective userid and operates with the third-party's MVS security authorizations. This proxy facility allows you to create Web transactions which access MVS resources which the end client would otherwise be denied access. NOTE: The third-party is not notified by the server that his MVS userid is being used as a proxy. Nor does any authentication procedure occur which involves the third-party, since password authentication is not performed. A third-party userid can be (mis) appropriated without the knowledge or consent of the owner. Indiscriminate or unsupervised use of the `RUNAUTH(xxxxxxxx)` keyword can lead to severe system-wide, security exposures. If the WWW master ruleset must be shared widely in your organization, we strongly suggests this facility be disabled.

Table 4–4. RUNAUTH Parameters and Values

Warning:
Because the RUNAUTH keyword is extremely powerful, server startup parameters are provided to limit the use of the RUNAUTH keyword. The WWWRUNAUTHLOCATIONS parameter can be set to disallow the use of RUNAUTH entirely, or to restrict its use outside the WWW master ruleset.

Specification of third-party userids with `RUNAUTH(xxxxxxxx)` can be disabled using the WWWRUNAUTHFORMATS start-up parameter.

RESOURCE (string)

The RESOURCE parameter/keyword specifies the generalized resource rule value to which the client userid must have read authorization before the URL can be accessed. If the client does not have this authority, the transaction is rejected by rescanning to the `SYSTEM/ERROR/403` (`Forbidden`) event procedure. The possible parameter values are:

Parameter Value	**Meaning**
RESOURCE(xxx)	When specified in the master ruleset, the value (a 1 to 39 byte string) replaces the transaction's current generalized resource value (if any). When specified in a subordinate ruleset, the value specified is appended to the transaction's current generalized resource entity value. An anomaly arises if a `RESOURCE( )` value is specified in a subordinate ruleset, but a value was not previously set by any master ruleset definition. This condition is handled by forcing the transaction's resource value to the subordinate ruleset name before the new value from the subordinate ruleset is appended. If the AUTHREQ keyword is not coded for a WWW rule that uses the RESOURCE keyword, the server assumes `AUTHREQ(YES)`.

Table 4–5. Resource Parameter and Values

Generalized resource authorization checks are performed before any transaction procedure is executed. Resource checking is performed if:

- Any non-blank value has been merged to the transaction's attributes during match processing.

Resource check is not performed if:

- The transaction's resource value is NULL (the RESOURCE parameter has not been specified for any of the matching rules).
- `AUTHREQ(NO)` is in effect for the Web transaction.

SSL(NO | COND | YES | LOCK | LOCKCOND)

The SSL parameter/keyword specify whether a Secure Sockets Layer (SSL) communications session must be in use before the transaction procedure can be executed. SSL should also be used at the client end to verify that the server being contacted is not impersonated by a third-party.

Note:
SSL encrypted connections should always be used by any transaction which sends or receives sensitive data.

How It Works

At the time the server selects a transaction procedure for execution, the server checks the SSL option. If an SSL connection is required, but an SSL connection is not in use, the server rejects execution by issuing an internal rescan to `SYSTEM/ERROR/SSL`.

The `SYSTEM/ERROR/SSL` procedure can be either a 403 (`Forbidden`) response or a 301 (`URL Has Moved`) response. Shadow OS/390 Web Server attempts to automatically reconnect the client's browser using SSL.

- When the server is configured for SSL support, an SSL connection is required for any transaction procedures where an SSL() option has been set into effect (except “NO”).
- If the server has not been configured for SSL support, transactions that have conditional SSL requirements, `SSL(COND)` or `SSL(LOCKCOND)`, are allowed to execute. Unconditional SSL requirements, `SSL(YES)` or `SSL(LOCK)`, cause the server to reject execution.

The possible SSL parameter values are:

Parameter Value	Meaning
SSL(NO)	(Default value) To override an SSL option specified by a previously matched-to `/*WWW` rule, you must explicitly specify `SSL(NO)`. When coded within a subordinate ruleset, `SSL(NO)` is ignored if a previous rule match has “locked-in” the requirement for an SSL connection.
SSL(YES)	When in effect, the option unconditionally requires the use of an SSL connection to the client. If the server is not configured for SSL support, no transaction procedure can be run. `SSL(YES)` can be overridden by an individual `/*WWW` rule. To disallow overrides, use `SSL(LOCK)`.

Table 4–6. SSL Parameters and Values

Parameter Value	Meaning
SSL(LOCK)	This is equivalent to `SSL(YES)`, except it cannot be overridden by a `/*WWW` rule defined in a subordinate ruleset. Use `SSL(LOCK)` on a generic rule in the master ruleset only for the use of SSL connections for a group of transactions. `SSL(LOCK)` can be overridden by another `/*WWW` rule which resides in the master ruleset. Do not code `SSL(LOCK)` except for master ruleset `/*WWW` rules.
SSL(COND)	If the server is configured to support SSL, then this option operates exactly like the `SSL(YES)` option. However, if the server is not configured for SSL support, transactions are allowed to execute without an SSL connection. Many of the `/*WWW` rules distributed with the server use `SSL(COND)`, rather than `SSL(YES)`. This allows new customers to operate before configuring SSL support in the server.
SSL(LOCKCOND)	If the server is configured to support SSL, then this option operates exactly like the `SSL(LOCK)` option. If the server is not configured for SSL support, transactions are allowed to execute without an SSL connection. If the server is started with SSL configured, you can use this option to “lock-in” SSL connection requirements.

Table 4–6. SSL Parameters and Values

Shadow OS/390 Web Server Subsystem Security

This section covers security processing issues related to the operation and administration of Shadow OS/390 Web Server subsystem with a view towards protecting the server and your MVS system from tampering. If you are using Shadow OS/390 Web Server only within the confines of your corporate Intranet, some of these issues do not apply.

Setting Limits for the Subsystem

Shadow OS/390 Web Server relies on the controlled transaction paradigm to protect your MVS system resources from unauthorized use. This means the server does not service any end user request without an explicitly defined transaction procedure. However, the server does run as an MVS APF-authorized started task with an inherent ability to carry out privileged operating system requests. This access should be denied to unauthorized programs.

The server’s security processing model explicitly defines a means of using third-party proxy userids for run-time authorization processing of Web transaction procedures. These userids can be set into effect without specifying a password, or any real-time authentication procedure which would make the owner of the userid aware of its use.

The highly sensitive nature of this type of processing is not uncommon to other OLTP subsystems, such as CICS, IMS, or JES. However, it does require enhanced security surveillance along with tight administrative control procedures to ensure that the facilities of the product are not misused to compromise your MVS System or some portion of the data residing there. We suggest that the following guidelines be followed:

Create a Separate Default RUNAUTH Userid

We strongly recommend that you:

- Create a unique userid to be used as the WWWDEFAULTRUNAUTH userid.
- Make sure the WWWDEFAULTRUNAUTH parameter is set during server start-up.

In the absence of an explicit WWWDEFAULTRUNAUTH specification, the server userid (and corresponding authorization level) is used for execution of all Web transactions, including those with public access. Because the server's userid must have write authority to Web transaction definitions, along with other security-related control values, you are leaving your system open if you do not create a new “default” userid.

As you configure the new userid, Shadow OS/390 Web Server assumes a low-level of authorization for it.

Configuring the New “Default” Userid

The primary use for the default userid is to provide a set of security subsystem permissions, under which fully public World Wide Web transactions are run. When configuring the default userid:

- Establish minimum permissions for the userid to allow operation of your public Internet applications. The default userid should not possess authority to access any MVS resource that you do not want shared with the entire Internet community.
- Do NOT grant write access to:
  - Any event procedure ruleset.
  - Any MVS load or procedure (such as, CLIST or TSO REXX) library. This includes signing-on to a TSO/E session. (Even with the recommended sign-on restrictions, the userid can still be used by the server to execute transaction procedures within one of the server maintained out-board TSO server regions.)

You must balance the limitations placed on the default userid against the level of administrative management exercised over Web transactions defined in subordinate rulesets. For example:

- When administrative management is tight, you can place a higher reliance upon the controlled transaction paradigm, because no transactions can be defined by developers which might damage the system.
- When administrative management is loose, you must limit the capabilities of this userid in order to protect against intentional or accidental misuse of the system. The more powerful the default userid, the more room there is to define Web transactions which could expose your MVS system to damage.

Note:
Files, which are shared or cached using DD name or DS name, and which specify the `SECURITY(SUBSYS)` attribute, can be accessed by the default userid. This happens because the subsystem's authority is used to open, read and close these shared datasets. All information within these files can potentially be distributed across the World Wide Web.

Associate the SWS Started Task With Its Own Userid

In addition to the Web transaction default userid, the Shadow OS/390 Web Server's started task is associated with some userid during the start-up process. We recommend that you:

- Always associate a UNIQUE userid with the Shadow OS/390 Web Server started task.
- Grant the unique userid the minimum authorization possible over any resources which directly belong to the Shadow OS/390 Web Server address space.
- Do not grant any authorization over other MVS resources which are not directly owned by the subsystem address space. Doing otherwise can provide a window of opportunity for malicious Web transaction developers.

Depending on the MVS security product installed on your system, if you set up a unique userid in this way, it can that ensure spin-off authorizations are not inadvertently assigned to the subsystem userid.

Limit the Subsystem's Access to Its Own Datasets

The subsystem userid should not have any more authorization to access its own datasets than it absolutely requires for proper operation. Nor should the subsystem's userid have write access to the product load library (which is APF-authorized), or to the user program library (from which high-level language Web transaction programs are fetched).

See the *Installation Guide* for the access level required by the subsystem's userid and datasets.

Protecting Subsystem Command and Control Interfaces

Use the Generalized Resource Rules

Define generalized resource lists to protect both the subsystem resources (such as, TRACEBROWSE, SEF, PARMS) and to provide limited access to URLs. Be sure the resource class name values are set for the subsystem's startup parameters RESOURCETYPE and URLRESOURCETYPE.

Be sure to limit access to the subsystem's “SEF” resource to developers responsible for implementing Web transaction definitions.

Protect the Subordinate WWW Rulesets

Limit access to the subordinate WWW rulesets to just transaction developers. Allow write access to only those libraries which an individual developer requires in order to define transactions for which the developer is directly responsible.

Provide Oversight and Administration for the Master WWW Ruleset

Because third-party proxy userids may be specified within the master ruleset, be sure to have tight control on write access to this library. You might want to only grant write access to:

- Security Administrative Personnel who are responsible for creating and maintaining generic WWW rules which govern security processing within each of the subordinate rulesets.
- System Programmers (occasionally/temporary) who need it to apply maintenance or upgrades to the Shadow OS/390 Web Server.
- Developers (Temporary) who tailor NEON Systems supplied Web definitions.

If the master WWW ruleset cannot be tightly controlled because of special needs within your organization, consider disabling the entire third-party proxy userid mechanism. To do this, set the parameter WWWRUNAUTHFORMATS to NOPROXY during server startup.

Limit Access to the Supplied /SWSCNTL URLs

Be sure the minimum requirement to execute the NEON Supplied URLs which are prefixed with `/SWSCNTL` is a valid MVS userid/password. If these transactions are unprotected, any outside user can exert control over the Shadow OS/390 Web Server's operational parameters.

Limit Access to Uncensored Trace Data

The server's wrap-around trace is a powerful diagnostic and auditing tool which can contain sensitive data by recording and displaying each step of WWW transaction execution, including many run-time data values.

This is indispensable when:

- Deploying new applications on the Web.
- Creating archival copies of the wrap-around trace as a permanent record of the interactions between the server, your application, and the end user.

Because new applications are generally developed with test data, the values recorded and displayed are usually not sensitive. They are, however, crucial to developers debugging new applications.

Warning:
In a production environment, trace records can contain sensitive data values which should only be visible to authorized personnel. The original developers of the application can be restricted from viewing the actual data values being processed, yet still be allowed access to a trace the logical flow of application executions.

For example, problems can occur when:

- Transaction definitions require a valid MVS userid and password for execution. The trace captures these values when the inbound HTTP request is recorded.
- You deploy a payroll application on the Web using Shadow OS/390 Web Server. The wrap-around trace facility records and can later playback sensitive data values such as an employee's social security number or salary.

Trace Browse Censorship Options

In order to provide censorship of the wrap-around trace, you must first define and activate the server's built-in generalized resource rules. (See the *Installation Guide*.) Censorship of the wrap-around trace information is based on:

- Users with **READ** access to both TRACEDATA and TRACEBROWSE resources can view uncensored data values and the underlying binary contents of trace records.
- Users with **READ** access to only TRACEBROWSE resource can view censored values. These users are denied access to the underlying binary trace contents.
- Users without access to either can not use the wrap-around trace facility.

The following server startup parameters control how wrap-around trace data censorship is applied:

Parameter	Default	Action
CENSORURLQUERYDATA	No	If set, all values derived from the HTML form's input fields are masked (that is, query data values sent as part of an inbound HTTP request). This censorship is applied to the trace of the inbound HTTP data stream, and to the trace of server created variables (traces activated by the PARSETRACE option).
CENSORURLAUTHDATA	Yes	User authentication data transmitted inbound as part of the Authorization: HTTP request header is censored.
CENSORHTTPRESP	No	Outbound response data is censored. This includes all data buffered for output using the SWSSEND API interface which is recorded when the SENDTRACE option is active. It also includes any individual data values inserted into an HTML skeleton by the HTML extension facility when the HTXTRACE option is active.
CENSORAPIDATAVALUES	No	Data values normally traced when the server's API tracing is active is censored. This includes the values of variables which are retrieved or set using the SWSVALUE API.
UNCENSORZOOMONLY	Yes	This controls how the trace appears to an authorized user. If set to YES, an authorized user has the same view of the trace as an unauthorized user (that is, records appear in their censored form). If set to NO, authorized users see uncensored information. In both cases, an authorized user can view the uncensored binary information. This option only affects how the text format display appears.

Table 4–7. Trace Browse Censorship Parameters and Values

Note:
Censorship of the wrap-around trace is applied to the text format view of the data, not to the actual data captured. The underlying binary trace records still contain potentially sensitive information. For this reason, you should restrict users from directly accessing the underlying VSAM Linear dataset in which the trace information is recorded, along with any datasets created to archive the wrap-around trace.

CHAPTER 5: Writing Web Transactions in REXX

Refer to either the *Shadow Programming Guide* or the online HTML for addition information regarding Shadow/REXX. Refer to the online HTML for information on Shadow REXXTOOLS by Open Software Technologies, Inc.

Shadow/REXX

The interpretive REXX language provides a powerful platform to create prototypes or low volume Web transactions. To create a REXX transaction, code a WWW event procedure with a REXX process section. This executes Shadow OS/390 Web Server's own Shadow/REXX interpreter which:

- Contains most of the facilities supported in standard REXX.
- Implements a pre-compiled step that speeds run-time execution.

Shadow OS/390 Web Server also includes a built-in interface to Shadow REXXTOOLS, which provides support for many add-on functions, such as the ability to access VSAM data sets.

/*REXX Process Sections

The `/*REXX` Process Section has its own set of keywords.

*/*REXX Statement Keywords*

The following keywords and operand values can be coded on the `/*REXX` process section header. Normally, you only want REXX procedures to be invoked when processing an actual event.

INIT

Parameter Value	Meaning
INIT(NO)	(default) This specifies that the REXX coding within the event procedure is not invoked when the encompassing rule is enabled.
INIT(YES)	This specifies that the REXX coding within the event procedure is invoked when the encompassing rule is enabled. When invoked for enablement, the event related variable, PHASE, is set to the value "INIT".

Table 5–1. INIT Parameter Values

PROC

Parameter Value	Meaning
PROC(NO)	This specifies that the REXX coding within the event procedure is not invoked when an actual event matches the event procedure criterion.
PROC(YES)	(default) This specifies that the REXX coding within the event procedure is invoked when an actual event is matched to the rule. When invoked for enablement, the event related variable, PHASE, is set to the value “PROC”.

Table 5–2. PROC Parameter Values

TERM

Parameter Value	Meaning
TERM(NO)	(default) This specifies that the REXX coding within the event procedure is not invoked when the event procedure is being disabled.
PROC(YES)	This specifies that the REXX coding within the event procedure is invoked when the event procedure is being disabled. When invoked for disablement, the event related variable, PHASE, is set to the value “TERM”.

Table 5–3. TERM Parameter Values

Coding the Process Section

Code Shadow/REXX procedures immediately after the `/*REXX` process section header statement. Shadow OS/390 Web Server reads and compiles the REXX code at the time the event procedure is enabled.

Note:
By pre-compiling the REXX code, errors can be detected which would not be seen by other REXX interpreters until execution time.

Shadow/REXX procedures can invoke external REXX-language subroutines, if they reside within:

- The same PDS dataset as the calling member.
- The SYSEXEC dataset.

Shadow/REXX Built-in Functions

Shadow/REXX implements all the standard REXX built-in function routines in addition to the standard REXX routines. The API index chart (online or in the *Shadow Programming Guide*) shows Web specific functions that you can use.

CHAPTER 6: File Serving Using Shadow OS/390 Web Server

URL Values and the UNIX File System

The UNIX file system is organized hierarchically with files stored within a directory or nested subdirectories (the file path). Individual file names appear in "dot notation" (`filename.filetype`) in which the first component is a unique file name, and the second is the file type value (such as, `.htm` or .txt). Most URL values are structured using this file naming convention, "`/path/../path/filename.filetype`".

Web servers have an intrinsic operation at their core around which all other functions (such as CGI and security processing) are built. Typically an inbound URL value references a file on the server which transmits something back to the client. For many Web transactions, the URL is the path and the file to be transmitted.

MVS File System

Native MVS file systems are not organized like the UNIX-based hierarchical files systems nor is the file naming convention the same. For this reason, Shadow OS/390 Web Server does not use the hierarchical method to process transactions. We did not implement a server that would require MVS-type file references for all URLs because:

- The URLs would look foreign from those implemented by all other servers.
- MVS lacks the concept of file type. That meant another method was still required to determine the type of data to be transmitted.

Because there was no intrinsic linkage, we defined alternate URL mapping. Shadow OS/390 Web Server matches URLs to rules, not files. This means, no MVS dataset can inadvertently be referenced, because none of the files are directly mapped. By using the controlled transaction paradigm, all gateways into the MVS system are closed until explicitly opened.

Note:
URL values are, in fact, tied to the MVS file system within Shadow OS/390 Web Server, but only indirectly. The URLs string defined (unless contained within the master WWW ruleset) are linked, by name, to the ruleset in which the transaction definition resides.

This places the security subsystem in control in which Web developers are allowed to define specific combinations of URL values by relating the URL value back to a ruleset owned by the developer.

Files Supported Directly by Shadow OS/390 Web Server

The server directly supports the use of MVS resident data that is contained within QSAM, BPAM, or PDSE datasets. External datasets can be referred to using either:

- A DD name, allocated to the server's address space, at either start-up time or dynamically any time thereafter.
- A fully-qualified MVS dataset name. The use aliases for these datasets is supported.

VSAM datasets are supported, in-directly, through the use of the Shadow REXXTOOLs VSAM Functions for REXX.

User written HLL transaction programs can access any type of MVS resident dataset using pre-allocated DD names, dynamically allocated datasets and one of the other MVS file organizations.

File Sharing and Caching

Shadow OS/390 Web Server contains facilities for sharing MVS PDS, PDSE, and sequential datasets across all transactions' subtasks. Sharing files has a number of advantages:

- Fewer CPU cycles are consumed because file open and close operations do not occur for each file reference.
- Information within the datasets can be cached so most file-based transactions require only a few milliseconds for execution.
- Files can be managed by the server according to parameter values instead of individually by each API referring to the same information.

File sharing is implemented as a part of all server API interfaces and not just for `/*FILE` process sections. Whenever a server interface references an MVS sequential dataset or a PDS member, the server attempts to obtain the information from a globally shared file using cached PDS directory entries or cached copies of the data itself, when its available. When a globally shared file is not accessible, each Web transaction uses a privately opened dataset copy.

When new `/*FILE` rules are accessed for the first time, a server start-up parameter allows all previously unseen files to be made globally shared dynamically. If you do not want this support, you can explicitly specify which datasets are to be shared or cached. The start-up parameters which control file sharing can be viewed at `/SWSCNTL/PARMS`.

Explicitly Shared Files

To designate files as explicitly shared include **DEFINE FILE** statements in the start-up parameterization REXX routine, SWSxIN00. When you explicitly designate shared files, the datasets are opened as part of the server initialization processing. You can specify individual caching and control options for each dataset.

Implicitly Shared Files

When a dataset is not designated during start-up processing as globally shared, you can do so during execution by setting either or both parameters, FILESHAREDSN and FILESHAREDDN, to YES. When a previously unshared file is accessed by a Web transaction for the first time and these options are set to yes, the dataset is globally shared dynamically.

How Shadow OS/390 Web Server Handles Files

Shadow OS/390 Web Server has a number of services which you can use to request outbound transmission of file resident data in response to a Web transaction. Chief among these is the `/*FILE` Process Section which is used within a WWW rule.

Each `/*FILE` section defines:

- The exact mapping between the inbound URL value and the MVS resident file to transmit.
- The relationship used to construct the correct response. For example, HTML forms are not transmitted outbound as GIF images.

The `/*FILE` section allows you to map either discreet or generic URL values to some entity within the MVS file system. You can define transactions which service the bulk of all file based requests before proceeding to use other services in developing specialized WWW applications.

Other server APIs allow you to reference and transmit files from within WWW rule REXX procedures, or HLL transaction programs. Or, you can use REXX and REXXTOOLs to create customized applications.

Building File Serving WWW Rules Using /*FILE

Whenever the WWW section has a `/*FILE` process sectionthe Web Server can:

- Map a discrete or generic URL value to an MVS dataset specification.
- Map requests to data contained within a sequential dataset (QSAM) or within a single member of a PDS or PDSE dataset (BPAM).
- Allow you to use a special form of `/*FILE` to store and transmit data from within the `/*WWW` rule.

Coding a /*FILE Process Section

This is an example of a `/*FILE` process section used within a WWW transaction rule. When executed, it transmits a GIF image to the Web browser from a PDS(E) library which was previously defined to the server.

Example

```
/*WWW /GIF/MYPICTURE
/*FILE DDNAME(GIFFILE) -
   MEMBER(JFFIMAGE) -
   CONTENTTYPE(image/gif) -
   FORMAT(BINARY) -
   DATATYPE(PDS)
```

To define a `/*FILE` process section, the `/*FILE` process section header statement must follow the `/*WWW` rule header statement.

This example references:

- The PDS dataset using a DD name (`‘GIFFILE’`).
- An explicit reference to the PDS member (`‘JFFIMAGE’`).

When the URL value `/GIF/MYPICTURE` is matched, and this rule is executed, the PDS member is transmitted to the client as MIME content type image/gif.

Because this example explicitly specifies all possible parameter values, the server can only map the URL value to the single PDS member.

/*FILE Transaction Operation

The transaction processing built-in to `/*FILE` and other server file APIs (see SWSFILE for REXX and SWSFILE for HLL Programs) handles the details of creating the correct response headers. Each execution of a `/*FILE` process section constitutes a whole and complete outbound transaction response.

The built-in transaction procedure handles the following items:

- Parses and uses portions of the inbound URL string to supplement explicitly coded keyword values. This allows you to create mapping between URL values and the file system.

- Substitutes a 304 (`Not Changed`) outbound response when the inbound transmission contains an “`If-modified-since:`” header and the file has not changed.

- Transmits only header information when the inbound transaction request is made using the HEAD HTTP method.

- Transmits a “`Last-modified:`” outbound header when the last modification data can be determined (this action is performed for PDS members only if ISPF statistics exist for the member).

- Processes HTML extension statements, if any, within the data file before transmission. These powerful Shadow OS/390 Web Server extensions can be used to tailor text data.
- Translates information for text format data from EBCDIC to ASCII under control of a national language mapping table. It strips extraneous blanks for each record and adds a CR character.

/*FILE Statement Keyword Syntax

When coding `/*FILE` statements:

- Place the `/*FILE` process section header statement immediately following the `/*WWW` rule header statement.
- If the `/*FILE` statement continues across multiple lines, indicate continuation by coding a trailing dash (‘`-`’) on all except the last header statement line.

The DATATYPE Keyword

Use the DATATYPE keyword to describe the basic transaction operation type performed. Allowable operands for the DATATYPE keyword are:

Keyword Operand	Transaction Operation Defined by This Operand
PDS	This specifies that the WWW transaction transmits a member of a PDS dataset to the Web client.
SEQ	This specifies that the WWW transaction transmits the contents of a sequential dataset to the Web client.
INLINE	This specifies that the transaction transmits data to the Web client which is present in the WWW rule. The transmitted data must immediately follow the `/*FILE` header statement. When `'INLINE'` is coded, at least one line of data must be present.

Table 6–1. DATATYPE Keywords and Their Meaning

Run-Time Overrides and Defaults

We strongly recommend that the DATATYPE keyword always be coded explicitly. If the keyword is omitted, the server assumes:

- `DATATYPE(INLINE)`, if data lines are present following the `/*FILE` section header.
- `DATATYPE(PDS)`, if no user data lines is present.

Note:
An `'INLINE'` data specification cannot be overridden at execution time because the `/*WWW` rule must be re-enabled to remove (or add) the `'INLINE'` data records.

If the wrong type of MVS dataset is referenced (such as referencing a PDS dataset when the definition specifies a sequential dataset, or vice versa), the server overrides this keyword at execution time to ensure the correct file organization is used.

The DDNAME Keyword

Use the DDNAME keyword to specify the MVS DD name of the PDS dataset (BPAM) or the sequential dataset (QSAM) from which data is obtained. Note that:

- The Web server does not perform dynamic allocation processing for user specified DD names.
- The DSNAME keyword cannot be coded if the DDNAME keyword is used.
- Do not code the DDNAME keyword when `DATATYPE(INLINE)` is specified.

The DD name must already be allocated to the Web server before the `/*FILE` transaction executes. You can do this by:

- Including DD statements within the start-up procedure.
- Allocating the DD statements from within another WWW rule using the dynamic allocation facilities of REXXTOOLs.

When the transaction is executed:

- If the coded DD name matches the DD name of a dataset that is being globally shared by the server, the shared dataset is used during processing.
- If the DD name is not being globally shared by the server, then the setting of the server's FILESHAREDDN start-up parameter is used to determine if the file can be shared globally. If it can, the server globally shares the dataset the first time the WWW transaction is executed.
- If the specified DD name is not currently allocated or no DD name or DS name value can be determined, the Web transaction is rejected with a “`URL Not Found`” error.

Run-Time Overrides and Defaults

If neither the DDNAME nor the DSNAME keywords are coded, the file processing transaction parses the matching inbound URL, and attempts to use a portion of it as a DD name value. The portion parsed out for use as a DD name reference is:

- **PDS.** The next-to-last blank delimited word of the matching inbound URL.
- **Sequential.** The last blank delimited word of the matching inbound URL.

See “Parsing URLs to Supply Missing /*FILE Keyword Values” on page 6-10 for a complete description of how the URL string can be used to specify a run-time DD name value.

The DSNAME Keyword

Use the DSNAME keyword to specify the fully qualified MVS dataset name from which data is to be obtained.

- You do not need to code quotation marks surrounding the DSNAME keyword operand.
- Do not code the DSNAME keyword when `DATATYPE(INLINE)` is specified.
- The DSNAME keyword cannot be specified if the DDNAME keyword is also present.

The Web server does not perform dynamic allocation of datasets with user specified DD names. DD name allocation must be done via the server start-up JCL or using other Web transaction processing facilities (such as REXXTOOLs Dynamic Allocation Interface).

When the transaction is executed:

- If the coded MVS dataset name matches the dataset name of a dataset which is being globally shared by the server, the shared dataset is accessed. Sharing occurs without regard to the DD name in use for the globally shared dataset.
- If the DS name is not being globally shared by the server, then the start-up parameter FILESHAREDSN determines whether the dataset is shared. If allowed, the server globally shares the named dataset the first time this Web transaction is executed.
- The transaction procedure allocates the MVS dataset to the Web server address space either permanently or temporarily, depending on whether the dataset is globally shared.

Run-Time Overrides and Defaults

If neither a DDNAME nor DSNAME specification is made, the server attempts to use a portion of the matching inbound URL value as a DD name specification, as described above for the DDNAME keyword.

The MEMBER Keyword

Only use this keyword for `DATATYPE(PDS)` transaction definitions. Use the MEMBER keyword to specify the 1-to-8 byte name of a member of the PDS dataset which is referred to by the DDNAME or DSNAME keyword.

The server transmits only the single member named to the Web client. If the member does not exist within the PDS dataset, the server rejects the WWW transaction with a “`URL Not Found`” message.

Run-Time Overrides and Defaults

If the MEMBER keyword is not coded for a PDS dataset, the server parses the matching inbound URL and attempts to use a portion of it as a member name. The characters which precede a period (if any) within the last blank-delimited word of the URL are used as the member name.

See “Parsing URLs to Supply Missing /*FILE Keyword Values” on page 6-10 for a complete description of how the URL string may be used to specify a run-time member name value.

The CONTENTTYPE Keyword

Use the CONTENTTYPE keyword to specify the MIME content type value to use in generating the HTTP response header. You can use any 1-to-50 byte MIME content type string, including values which are not defined within the server’s internal MIME table.

Coding the Keyword

When you code this keyword, any string specified is considered to be a valid HTTP MIME content type string by the server if it contains a “/” character. Once the server finds an embedded “`/`” character, it performs no further validations upon the operand value, but simply transmits the specified value as part of the outbound response header.

Specifying File Extensions

Instead of coding an HTTP MIME content type string, you can specify a 1-to-8 byte file extension value, such as ‘`TXT`’, ‘`HTM`’, or ‘`HTML`’. The server converts standard file extension names into the appropriate MIME type.

The server performs this conversion by determining if the string contains a “`/`” character. If it does not, the server assumes a file extension value was coded and references the MIME Table to find the corresponding MIME content type string.

If the value is not a known file extension name, the server assumes ‘`TXT`’ and yielding a MIME content type of ‘text/plain’.

Run-Time Overrides and Defaults

If the execution time content type was not derived from any other source, the server parses and attempts to use a portion of the matching inbound URL as a standard PC-LAN file type designation. This designation, if present, is in turn transformed to a MIME content type value.

The portion of the URL which is parsed is any characters which follow a period within the last blank delimited word of the inbound URL.

See “Parsing URLs to Supply Missing /*FILE Keyword Values” on page 6-10 for a complete description of how the URL string may be used to specify a run-time content type value.

The FORMAT Keyword

Use the FORMAT keyword to specify the basic data contents of the file data sent. Valid operands for this keyword are:

Keyword Operand	Contents of File Data and Server Processing
BINARY	Binary format denotes information which the server transmits to the Web client as is, without modification. The data can be an encoded object, such as a Graphics Interchange File (`GIF`), or text data which is stored in ASCII with a trailing carriage return character at the end of each line (`X'0A'`). In most cases, this data was placed onto the MVS (perhaps using FTP), and is unintelligible to MVS. It is extremely difficult to properly store binary format data within a WWW event procedure as `‘INLINE’` data. We strongly recommend that you not attempt to do so.
TEXT	Text format denotes information which the server automatically translates from EBCDIC to ASCII before transmission. Trailing blanks are removed from each line and a carriage return character is appended to each line before transmission. These files usually originated on the MVS system and would be unintelligible to a Web browser without this translation.

Table 6–2. FORMAT Keyword Operands

Run-Time Overrides and Defaults

If the format of the data is not specified using this keyword, the server attempts to derive the expected file contents format by checking the MIME content type against the MIME table. The table contains a list of anticipated file content formats matching each of the known MIME types and file extension values.

The HTX Keyword

Only use the HTX keyword when the file contents is `FORMAT(TEXT)`. The HTX keyword is invalid if `FORMAT(BINARY)` has been specified explicitly for the `/*FILE` statement.

The HTX keyword specifies if or how HTML Extension Statements are processed before the contents of the file (which are altered by such processing) are transmitted. Valid keyword operand values are:

Keyword Operand	Operation Of Transaction Processor
HTX(NO)	This surpresses the HTML Extension Statement processing of the file data. The server transmits the information without alteration
HTX(YES)	(default) HTML Extension processing is normally performed for all text members.
HTX(REXXRULES)	When HTML extensions are processed, if an uninitialized variables is encountered, the server replaces the uninitialized variable with a NULL string. However, you can code to override the normal processing of uninitialized variables. When this processing is requested, uninitialized variables are replaced with the uppercase name of the variable itself (similar to REXX interpreter processing)

Table 6–3. HTX Keyword Operands

Parsing URLs to Supply Missing /*FILE Keyword Values

The built-in `/*FILE` transaction processor attempts to use portions of the matching inbound URL to supply values that are missing at execution time. This allows you to create `/*FILE` WWW rules that operate generically by mapping a part of the URL to the MVS file specification.

This section covers how the server implements the URL parse and then uses the portions derived. The steps in parsing the URL are:

1. The server begins the URL parse by replacing all “`/`” characters within the matching URL with blanks. It then divides the URL into a series of blank delimited words.
2. The last blank delimited word of the URL string is examined to see if it contains an embedded period. If it does, the last word is split into two separate words, at the point where the period occurs.
3. After this parsing operation, the server processes two or three portions of these elements, depending on whether the `/*FILE` rule refers to a PDS or a sequential dataset. Any portion, which is not present, is assumed to be blank.

Example

Given an example URL string, “`/NEON/HTMFILE/FILESECT.HTM`”, the server derives and maps portions of the URL as follows:

Description of URL Segment	Parsed Segment of URL	Use for PDS Dataset References	Use for Sequential Dataset References
Any segment(s) preceding the next-to-last blank-delimited word located in step 1	NEON	Not Used	Not Used
The next-to-last blank-delimited word located in step 1	HTMFILE	Used as DD name when DDNAME and DSNAME keywords are omitted	Not Used
The portion of the URL preceding a period, as described in step 2	FILESECT	Used as PDS member name when MEMBER keyword is omitted.	Used as sequential file DD name when DDNAME and DSNAME keywords are omitted
The portion of the URL following a period, as described in step 2	HTM	Used as a file extension and converted to a MIME Content Type when CONTENTTYPE keyword is omitted. File FORMAT value can also be derived, if not otherwise specified.	Used as a file extension and converted to a MIME Content Type when CONTENTTYPE keyword is omitted. File FORMAT value can also be derived, if not otherwise specified.

Table 6–4. Example of How the Server Maps “/NEON/HTMFILE/FILESECT.HTM”

When any portion of a parsed URL is used by the transaction processor, it must:

- Contain characters that are valid for MVS DD names or PDS member names.
- Be the correct length.

File extension values can be 1-to-8 bytes in length and must be a value present within the MIME Table.

MIME and File Extensions Table

Shadow OS/390 Web Server contains a table that relates common PC-type file extension values to recognized MIME (Multipurpose Internet Mail Extensions) types and vice versa.

This table is used by `/*FILE` sections when default PDS processing is used to determine a PDS member name and the member format (binary or text), or output content type by parsing values from the URL.

Two types of look-up operations are performed:

- An attempt to use a 2 to 8 byte file extension value (for example, html) to obtain a matching MIME content type value
- An attempt to use the MIME content type value to determine if the source member is in binary or text format

Type of Match	What It Does
No Match (File Extension)	When a file extension look-up operation is performed and no match is found, the Web server assumes a file extension value of "`.txt`". This yields a MIME type of "text/plain" and an input data format of text.
No Match (Content Type)	When a MIME content type name look-up is performed and the content type value is unmatched, Shadow OS/390 Web Server assumes an input data format of binary, and uses the content type value as specified.
Questionable	If in doubt, the server uses the presence or absence of an embedded "`/`" (slash) character to determine if a file extension look-up or a content type name look-up is being performed. All valid content type values have an embedded slash character.

Table 6–5. MIME Matches

Each entry in the following table shows the file extension value that the Web server recognizes, and the "Content-type:" value that is used for outbound transmission. The final column shows the storage format which the processor assumes for a data set member.

File Extension	MIME Content Type	Storage Format
.htm .html .mdl	text/html	Text
.txt	text/plain	Text
.jpg .jpe .jpeg	image/jpeg	Binary
.gif	image/gif	Binary
.gz	application/x-gzip	Binary
.z	application/x-compress	Binary
.pac	application/x-ns-proxy-autoconfig	Binary
.jf .ls .mocha	application/x-javascript	Binary
.tcl	application/x-tcl	Binary
.csh	application/x-csh	Binary
.ai .eps .PS	application/postscript	Binary
.exe .bin	application/octet-stream	Binary
.cpio	application/x-cpio	Binary
.gtar	application/x-gtar	Binary
.tar	application/x-tar	Binary
.shar	application/x-shar	Binary
.zip	application/x-zip-compressed	Binary

Table 6–6. Only Shows File Extensions that Shadow OS/390 Web Server Recognizes

File Extension	MIME Content Type	Storage Format
.sit	application/x-stuffit	Binary
.hqx	application/mac-binhex40	Binary
.avi	video/x-msvideo	Binary
.qt .mov	video/quicktime	Binary
.mpeg .mpg .mpe	video/mpeg	Binary
.wav	audio/x-wav	Binary
.aif .aiff .aifc	audio/x-aiff	Binary
.au .snd	audio/basic	Binary
.fif	application/fractals	Binary
.ief	image/ief	Binary
.bmp	image/x-MS-bmp	Binary
.rgb	image/x-rgb	Binary
.ppm	image/x-portable-pixmap	Binary
.pgm	image/x-portable-graymap	Binary
.pbm	image/x-portable-bitmap	Binary
.pnm	image/x-portable-anymap	Binary
.xwd	image/xwindowdump	Binary
.xpm	image/x-pixmap	Binary
.xbm	image/x-bitmap	Binary
.ras	image/x-cmu-raste	Binary
.tiff .tif	image/tiff	Binary
.texi .texinfo	application/x-texinfo	Binary
.dvi	application/x-dvi	Binary
.latex	application/x-latex	Binary
.tex	application/x-tex	Binary
.rtf	application/rtf	Binary

Table 6–6. Only Shows File Extensions that Shadow OS/390 Web Server Recognizes

Inline File Processing

Inline processing means the `/*FILE` process section of a WWW event procedure contains the data to be transmitted to the Web client. For example:

```
/*WWW /SAMPLEDATA
/*FILE
<HTML>
<BODY>
<P>
```

This is a sample inline HTML data stream.

```
</BODY></HTML>
```

The HTML data stream transmitted to the Web client is contained within the WWW event procedure. Because `'INLINE'` is the default value if any data records follow the `/*FILE` section header statement, it was not necessary to code `DATATYPE(INLINE)` as a keyword.

Only the `DATATYPE(INLINE)`, CONTENTTYPE, FORMAT, and HTX keywords can be coded for `'INLINE'` data transmission.

Note:
We strongly recommend that you only use `'INLINE'` data streams for data formatted as TEXT.

Examples of /*FILE Rules

The following examples show various methods in which `/*FILE` process section rules can be defined, which include different types of access and level of administrative control.

Sequential Example 1

For the following Web transaction rule:

```
/*WWW /SEQDATA/*
/*FILE DATATYPE(SEQ)
```

If the inbound URL value is `'/SEQDATA/GIF.TIF'`, the transaction attempts to send the sequential dataset allocated to the GIF DD name. A MIME content type of ‘image/tiff’ is generated in the outbound response. The content format is implied to be binary, so no HTML extension processing is performed.

If the inbound URL value is `'/SEQDATA/GIF/'`, then the transaction attempts to send the sequential dataset allocated to the GIF DD name. A MIME content type defaults to ‘text/plain’ because no other specification was made or located either by the URL parse or by a keyword operand.

Sequential Example 2

For the following Web transaction rule:

```
/*WWW /SEQDATA/*
/*FILE DATATYPE(SEQ) -
     DDNAME(MYFILE)
```

If the inbound URL value is `'/SEQDATA/ABC.TIF'`, the transaction attempts to send the sequential dataset allocated to the `MYFILE` DD name (`'ABC'` is not used because the DD name is explicitly specified in the rule).

A MIME content type of ‘image/tiff’ is generated based on the input URL parse of the ‘`.TIF`’ extension. Because the content format is implied to be binary, no HTML extension processing is performed.

Sequential Example 3

For the following Web transaction rule:

```
/*WWW /SEQDATA/*
/*FILE DATATYPE(SEQ) -
      DSNAME('CSD.AI38.S.TEXTDATA') -
      CONTENTTYPE(text/plain) -
      FORMAT(TEXT)
```

Any inbound URL value which begins with the string `'/SEQDATA/'`, transmits the explicitly specified file as MIME content type ‘text/plain’. Because the content format is implied to be text, HTML extension processing is performed.

PDS Example 1

For the following Web transaction rule:

```
/*WWW /GIF/*
/*FILE DATATYPE(PDS) -
     DDNAME(GIFFILE) -
     CONTENTTYPE(image/gif) -
     FORMAT(BINARY)
```

The inbound URL is parsed to yield the unspecified PDS member name value. The member is transmitted as MIME Content Type ‘image/gif’. Because the content format is binary, no HTML extension processing is performed.

For the URL value, `'/GIF/XYZ.ABC'`, the XYZ member of the PDS is transmitted.

PDS Example 2

For the following Web transaction rule:

```
/*WWW /DOCDATA/*
/*FILE DATATYPE(PDS) -
      DDNAME(DOCFILES)
```

Each inbound URL value is parsed to yield a PDS member name and file extension value. The specified member is sent using a MIME Content Type derived from the file extension value. If no file extension is parsed from the URL, the default used is ‘text/plain’. HTML extensions are processed when the MIME Content Type indicates a text format member.

The URL, `‘/DOCDATA/FILESECT.HTM’` transmits the `‘FILESECT’` member as ‘text/html’.

The URL, `‘/DOCDATA/FILEPICS.JPG’` transmits the `‘FILEPICS’` member as ‘image/jpeg’.

CHAPTER 7: HTML Extension Facility

Shadow OS/390 Web Server supports a run-time facility that allows you to create customized HTML forms and Web pages in response to run-time execution conditions. By including HTML-like statements in your source file, Shadow OS/390 Web Server evaluates the HTML statements and customizes the response either the client's request or the current operational state of the server. More powerful capabilities are available when this operation is combined with other server processing facilities, such as
`/*EXECSQL`.

Using the HTML Extension Facility

The following describes the functions and limitation available.

Insert Variable Text Into the Output

Run-time event related variables, global variables, server provided information values, or merge processing data (such as DB2 result set column data) can be inserted into a text member before it is transmitted to the end user. These text insertion extention statements specify what run-time value should be placed in the output text stream. That value replaces the original HTML extension statement.

HTTP Response Control Statements

HTML extensions can be used to generate page-specific HTTP response header information. The API, SWSRESP, allows rules to generate HTTP response headers. See the *Shadow Programming Guide* for more information.

Conditional Statements

Conditional expressions can be coded as HTML extension statements using `<%if%>`, `<%else%>`, and `<%endif%>` statements. Conditional statements allow you to select which group(s) of lines within a text member are actually transmitted. The selection is based on run-time evaluation of a condition.

Iteration Statements

One or more text lines, delimited by iteration statements, can be repetitively processed during evaluation. The number of repetitions is controlled by run-time evaluation of a variable value.

Repetitive line groups are used primarily in conjunction with the `/*EXECSQL` merge processing interface.

Other Control Statements

Other HTML extension directives allow you to control processing during a file tailoring operation. For instance, you can abort file tailoring, or cause the DB2 result set row cursor to be advanced.

Merging Data From Other Server Facilities

The server implements specialized merge processing for some built-in facilities, such as `/*EXECSQL`. These special interfaces allow you to tailor output streams using variable information available only through the built-in component. For instance, when file tailoring is invoked by a `/*EXECSQL` components, the interface makes each DB2 column available to the HTML extension processor, along with other information pertaining to the execution of the SQL request.

These special interfaces provide an extremely powerful, flexible, turnkey formatting capability.

Scope and Limitations of the Facility

HTML extension statements are coded within a text member, just like any other HTML tag. The HTML extension processor uses escape delimiters, which are similar-to, but not part of, the HTML tag language. These escape delimiters enclose all HTML extension statements, denoting them as separate from the rest of the source file text.

During transmission processing, all HTML extension statements are removed from the source file and evaluated. Text insertion statements are replaced within the source file by the evaluated variable or data item to be inserted.

Processes for Text-type Files Only

The key criterion used by Shadow OS/390 Web Server in whether HTML extensions are processed is the format of the source file and the transmission mode. HTML extensions are processed only within text format data members, never binary, nor can the HTML extensions be processed when the output transmission operation is binary.

Processes for All Text Files - Regardless of MIME Type

Extensions are processed within all text format data files, regardless of the MIME content type being used to transmit the data. HTML extensions are recognized and processed within a data member, even if the member is transmitted to the Web client as some type other than “text/HTML” or “text/plain”.

HTML Extension Processing is Enabled by Default

By default, extension processing is enabled for all text format transmissions. To suppress processing of HTML extension statements, you must use keywords to explicitly request that it is not performed.

Supports for All Web Server File Interfaces

Extension statement recognition and processing occur for all transmission operations initiated by:

- `/*FILE` process sections.
- `/*EXECSQL` process sections.
- The SWSFILE API Function.
- The HLL File API Interface.

Pre-Compiled for Execution-Time Efficiency

For source file data retained in cache by the server, the HTML extension facility generates and caches an internal p-code array during the first outbound expansion/transmission request. This speeds evaluation for subsequent expansion/transmission operations because the source file does not need to be re-scanned.

Limits on Various Syntactical Elements

The chief priority of the server development team is to ensure that the server uses the smallest amount of CPU time and other machine resources. Because the HTML extension statement processing occurs frequently during server operation, it performs many text scan/replace operations, which by their very nature, consume CPU cycles.

We have made the HTML extension processor as efficient as possible by conforming to various Web server internal architectural conventions, such as avoiding extra CPU time expenditures that would otherwise be required to handle the few special cases of especially large or complex source files.

The HTML extension processor imposes the following limitations on source file data contents and evaluation time processing:

- No single variable value longer than 16000 bytes in length.
- No individual text line can exceed 32000 bytes in length.
- ‘If’ statements cannot be nested within encompassing ‘if/endif’ statements more than 10 levels deep.
- ‘Do’ groups cannot be nested within encompassing ‘do/enddo’ groups more than 10 levels deep.
- No more than 30 statement labels can be present within a single source file.
- No more than 20 pairs of HTML statement escape delimiters (‘`<%`’ ... ‘`%>`’) can be present on a single source line and no more than 1 pair for all statement types except insertion statements.
- During pre-scan, before evaluation, no more than 30 unresolved ‘leave’ statements can be remembered before the matching ‘enddo’ statement is scanned.

This allows enough space for a total of approximately 550 HTML extension statements to be coded within any one source file. Extension processing fails if too many statements are present within a single source file.

Rules for Coding HTML Extension Statements

HTML extension statements are coded within a source file just like any other HTML tag, except these delimiters are not part of the HTML tag language. The following covers the rules for coding HTML extension statements.

HTML Extension Statement Escape Delimiters

Escape delimiters must enclose all HTML extension statements. This keeps the statements separate from the rest of the source file text. Each HTML extension statement must:

- Begin with a '`<%`' delimiter.
- End with the '`%>`' delimiter.

During evaluation/expansion processing, the entire statement (including the delimiters) is removed from the source file before output. Executable statements are evaluated and processed to perform the action(s) indicated. Insertion statements are replaced by the run-time value of the statement operand.

Escape Delimiters Recognized In All Locations

The opening and closing delimiters (and the extension statement within it) are recognized regardless where they appear in the source file. The HTML extension facility processes the extension statement delimiters when they are coded:

- Within quoted text.
- Inside a regular HTML markup tag.
- Within an HTML comment markup.

If you need to actually transmit the character strings, '`<%`' or '`%>`', you must use the HTML entity reference mechanism to specify these as text values. For example:

- Code '`&lt;%`' in place of the opening delimiter value '`<%`'.
- Code '`%&gt;`' in place of the closing delimiter value '`%>`'.

Use Escape Delimiters to Add Comments

HTML Extension Statement Comments must be wholly contained on a single text line by themselves, with no other leading or trailing non-blank text. Each HTML extension comment:

- Begins with '`<%!`' delimiter.
- Ends with '`%>`' delimiter.

Text within these delimiters, and the delimiters themselves, are removed when the HTML extensions are processed.

In this example, only lines 000001 - 000006 are comments.

```
***** **************************** Top of Data***************************
 000001 <%!******************************************************************%>
 000002 <%! Note that this text document contains Shadow OS/390 Web Server HTML
*%>
 000003 <%! Extensions and is used to dynamically generate 4 different        %>
 000004 <%! pages, depending on which query variables and cookies are        *%>
 000005 <%! transmitted on the in-bound request.                             *%>
 000006 <%!******************************************************************%>
 000007 <HTML>
```

No Continuation of Statements

Each HTML extension statement must:

- Be entirely contained on a single input text line; extension statements cannot be continued across multiple lines.
- Have the same number of opening and closing delimiters.

Single Statement Per Source Record (Except Text Insertions)

For all HTML extension statement types, except text insertions, the statement must be coded on a source file line with no other non-blank text preceding or following it. Only one statement can appear on a source line.

Text insertion type statements can be located anywhere, even intermingled among other source file text. Multiple insertion type statements can be present on a single source file input line.

Text insertion statements can be nested within other statements to de-reference variable data.

Mixed Case Coding Allowed

HTML extension statements can be coded in upper, lower, or mixed case.

Reserved Words Not Valid as Variable Names

The following words are reserved. They cannot be used as the names of variables or as statement labels. Nor can they be inserted into an HTML extension statement at run-time using nested insertions statements.

if	leaves	abort	ge
else	next	contains	lt
endif	advance	eq	le
do	noadvance	ne	
enddo	exit	gt	

Using Statement Operands

Each operand of an HTML extension statement can be one of the following:

- A numeric literal
- A string literal
- A reference to one of the variable types
- A pseudo-function call

When variables are referenced, the maximum variable name size that can be used in an HTML extension statement is 50 bytes.

Numeric Literals

- Are coded as an integer value with or without a leading minus sign. The range of values is limited to plus or minus 2,147,483,647.

- Can be used as operands of `<%do%>` and `<%if%>` statements; however, they cannot be used as text insertion statement operands.

String Literals

- Are coded as a sequence of bytes enclosed within single or double quotation marks. The maximum length supported is 256 bytes.

- Can be used as operands of `<%do%>` and `<%if%>` statements; however, they cannot be used in text insertion statements.

REXX Dynamic Variables

- Can be referenced by name as a statement operand. The server obtains the current value of the REXX variable from the REXX variable pool using the IRXEXCOM interface.

- Are only accessible when the file transmission operation is invoked by a call from REXX to the SWSFILE API function. If you attempt to reference a REXX language dynamic variable value outside this environment, the variable is uninitialized/undefined.

WWW Event-Related Variables

The name of one variable (created for each inbound Web transaction event) can be specified as an operand. For example, `WWW.USERID`

Global Variables

The name of one variable can be specified as an operand. For example, `GLOBAL.COMPANY.NAME`

GLVEVENT Temporary Variables

The name of one variable can be specified as a statement operand. For example, `GLVEVENT.USER.VALUE`

- No variables of this type exist until explicitly initialized by a Web transaction procedure.
- '`GLVEVENT.`' variables are the only type of task level variable that can be created by HLL programs for use in HTML extension processing.

Built-in HTXINDEX Variable

The special variable names, HTXINDEX or HTXINDEX.label, can be referenced as statement operands. HTXINDEX variables only exist while HTML extension processing is underway.

An HTXINDEX variable:

- Is always evaluated as an integer value that matches the iteration count of a `<%do%>` group.
- References the counter of the inner-most `<%do%>` group currently being executed.
- If no group is running, the variable references the final count value of the most recently ended group. If no group has ever been executed, the referenced value is zero.

A variable in the form, "HTXINDEX.label", refers to a specific `<%do%>` group iteration counter by name. The "label" reference must match the label name coded for a `<%do%>` statement within the source file. If this group hasn't been executed, the referenced value is zero. If the group has terminated, the final iteration count value for the named group is returned.

Special Merge Processing Variables

When HTML extension processing is invoked automatically by other Shadow OS/390 Web Server components, special variables are made available to the interface for merge processing. These variables are only available when HTML extension processing is invoked indirectly by certain Web server components, such as `/*EXECSQL` process section definitions.

“Merging Data From Other Server Facilities” on page 7-2 for a description of the special variables available to each interface.

SWSINFO Pseudo-Function Call

You can specify a pseudo-function call upon the SWSINFO built-in function as the operand of an HTML extension statement by using the following format:

```
SWSINFO(item)
```

The value of item can be any of the character strings used as an input argument for the SWSINFO built-in function. (Do not put quotation marks around the operand.)

Run-time Operand Evaluation

The following rules govern run-time evaluation of HTML extension statement operands.

Uninitialized Variable References

If the requested value of a variable cannot be obtained because the variable is uninitialized, or not available within the environment from which the output operation was invoked, the evaluated value of the operand depends on options set when the HTML extension facility was invoked.

The default action is to substitute a NULL string in place of the variable. You can request that uninitialized variables be replaced with the uppercase name of the variable itself, but you must explicitly request this type of handling. This algorithm corresponds to the way in which REXX handles undefined/uninitialized variables.

REXX Type Stem Variables Not Resolved By HTML Extension Facility

Each variable name operand must be a fully resolved name. The interface does not support the use of stem variables (as in REXX), where each portion of the variable name is evaluated before the final variable name is derived and evaluated.

For example, assuming the following values were set up for REXX variables:

```
B.A = "CDE"
B.2 = "ABC"
A = '2'
```

If `B.A` is used as an HTML extension statement operand, the evaluated result is the string ‘`CDE`’, not the string ‘`ABC`’.

Insertion Statements May be Nested

HTML extension text insertion statements can be nested within other extension statement types. Nested statements are resolved at run-time from the inner-most to the outer-most nesting level.

- A nested insertion statement cannot be used to specify the overall statement operation type (such as ‘if’ or ‘do’), nor can it use any other HTML extension statement reserved word.
- A nested insertion statement can be used to parameterize an HTXINDEX reference. For example, “`HTXINDEX.<%var1%>`” where “`var1`” contains a valid label name string.
- All other label name operands references cannot be made using a nested insertion statement. For example, “`<%leave <%var1%>%>`”, where “`var1`” is intended to supply the label name, is not a valid use of nesting.
- Because nested insertion statements must be completely resolved during output processing, and cannot be pre-compiled, they require somewhat more CPU time for processing than non-nested operand references.

An example of operand nesting

1. Assume the run-time variable, “`A`”, is set to the value “`2`”, and that the variable “`B.2`” is set to the value “`ABC`”.
2. Assume the following nested reference is coded in a source file being evaluated by the HTML extension Facility:

   ```
   <%B.<%A%>%>
   ```

   The facility resolves the inner-most expression, ‘`<%A%>`’, as ‘`2`’, yielding the intermediate string ‘`<%B.2%>`’. This intermediate expression is evaluated as ‘`ABC`’, which is the value of the variable ‘`B.2`’.

HTML Extension Text Insertion Statement

A text insertion statement causes the statement operand to be evaluated by the extension facility. The evaluated value replaces the entire text insertion statement.

Statement Syntax

An insertion statement consists of a single operand reference enclosed by extension statement delimiters. During processing, the entire statement is replaced with the value of the operand; numeric and string literals cannot be used as text insertion statement operands.

Code text insertion statements as follows:

```
<%operand%>
```

Here, a single operand reference value, '`operand`', is coded. At evaluation time, the value of operand replaces the entire HTML text insertion statement.

Refer to:

- Statement operands for a list of supported HTML extension statement operand types.
- Run-time operand evaluation for details about uninitialized variable handling and nesting of insertion statements.

Substitution Example

The following table shows the actual source file text used to specify a text insertion operation and the resultant output after the substitution has been performed. The substitution operation actually takes place when the page is retrieved by the server.

HTML Text Insertion Operation Example	
Text Within This File	The User Agent value for the browser you are currently using is '`<%WWW.USER_AGENT%>`'.
Results After Substitution	The User Agent value for the browser you are currently using is `'  '`

At run-time, the value of the WWW.USER_AGENT event related variable is substituted for the HTML extension escape sequence. The second row of the table should contain the User Agent value transmitted by your Web browser.

HTML Extension Run-time Condition Checking

You can tailor source file output by testing conditions at run-time using the HTML extension statements `<%if%>`, `<%else%>`, and `<%endif%>`.

<%if%> Statement Syntax

The '`<%if%>`' statement tests a run-time condition and repositions source file evaluation based on whether the condition is true or false.

Each '`<%if%>`' statement consists of:

- The reserved keyword, 'if' (in either upper or lowercase).
- Two operand values to be compared against each other.
- Each operand separated by a condition keyword.

The '`<%if%>`' statement is coded as follows:

```
<%if oper1 condition-keyword oper2%>
```

- The ‘if’ statement must appear as the first blank delimited word in the HTML extension statement.
- Both ‘oper1’ and ‘oper2’ must be present. See “Using Statement Operands” on page 7-6 for more information.

The ‘condition keyword’ must be one of the following reserved words:

Condition Keyword	Description
EQ	operand 1 equal to operand 2
NE	operand 1 not equal to operand 2
LT	operand 1 less than operand 2
LE	operand 1 less than or equal to operand 2
GT	operand 1 greater than operand 2
GE	operand 1 greater than operand 2
CONTAINS	operand 1 contains the string operand 2

Table 7–1. IF Statement Condition Keywords

- Each ‘if’ statement in the source file must have a matching ‘endif’ statement.
- If both ‘oper1’ and ‘oper2’ can be evaluated as integral values, the comparison is performed using integer values rather than a string comparison.
- When the ‘contains’ condition is evaluated, the search for an occurrence of ‘oper2’ within the ‘oper1’ value is conducted as a string comparison. The search is conducted as a caseless comparison by folding both operands to uppercase.

<%else%> Statement Syntax

The ‘`<%else%>`’ statement allows you to delimit source statements to be evaluated when the preceding ‘if’ statement is false. Each ‘else’ statement consists of:

- The reserved keyword ‘else’ (in upper or lowercase),

  The ‘`<%else%>`’ statement is coded as follows:

  ```
  <%else%>
  ```

  ‘else’ can only be specified within a ‘if/endif’ statement group.

<%endif%> Statement Syntax

The ‘`<%endif%>`’ statements marks the end of source statements evaluated by a preceding ‘if’. Each ‘endif’ statement consists of:

- The reserved keyword ‘endif’ (in upper or lower case),

  The ‘`<%endif%>`’ statement is coded as follows:

  ```
  <%endif%>
  ```

  ‘endif’ can only be used to terminate a preceding ‘if’ statement group.

Condition Statement Example

The following shows the use of a condition statement to test the value of query variables:

Example

```
<HTML>
<BODY>
<p>
<%if www.field.0 EQ 0%>
    No query variables were sent with this URL.
<%else%>
    <%www.field.0%> query variables were sent with this URL.
<%enddo%>
</BODY>
</HTML>
```

HTML Extension Iteration Statements

One or more source file lines can be grouped and repetitively evaluated a specified number of times by using the following HTML extension facility statements:

- The ‘`<%do%>`’ statement initiates the iteration loop and specifies an iteration count.
- The ‘`<%leave%>`’ statement is coded within the group to cause evaluation processing to exit the group.
- The ‘`<%next%>`’ statement is coded within the group to cause evaluation processing to skip to the top of the group and begin the next (if any) iteration.
- The ‘`<%enddo%>`’ statement marks the end of the source lines included within the iteration group.

Using Named Iteration Groups

If you label an iteration group, you can:

- Explicitly specify which group a ‘leave’, ‘next’, or ‘enddo’ statement refers to without ambiguity.
- Refer to the group's iteration count value by name, using an ‘HTXINDEX.label’ reference.

A label name is assigned to an iteration group by defining the label within the HTML extension ‘do’ statement which inaugurates the group. All label names must:

- Be 1 to 8 characters in length.
- Begin with an alphabetic character.
- Be either alphabetic or numeric after the first character.

If ‘do’ groups are nested and no label names are defined, ‘leave’ or ‘next’ statements refer to the inner-most group. When label names are assigned, ‘leave’ and ‘next’ statements can refer to outer groups by name using the ‘HTXINDEX.label’ operand. The simple form ‘HTXINDEX’ always references the inner-most iteration count.

<%do%> Statement Syntax

The HTML extension ‘`<%do%>`’ statement begins each iteration group and specifies, by operand, the number of times the group should be repeated.

Each ‘`<%do%>`’ statement consists of:

- An optional label name for the iteration group.
- The reserved keyword, ‘do’ (in either upper or lowercase).
- An operand reference which provides the iteration count limit as a positive integer value.
- An optional keyword that indicates automatic cursor advance (used for `/*EXECSQL` merge processing).

Use the following format to code the statement:

```
<%label: DO oper1 ADVANCE%>
```

The statement label is optional, but if coded, it must:

- Be 1 to 8 alphabetic or alpha-numeric characters.
- Have a colon at the end of the label name.
- Have at least one blank after the colon to terminate the label name.

The reserved word, ‘do’, must appear as the first blank delimited word within the HTML extension statement, or as the second word when a label name is present.

The iteration count limit, ‘oper1’ follows the ‘do’ keyword. The operand can be of any type discussed in “Using Statement Operands” on page 7-6. The operand is evaluated at run-time to obtain the iteration count limit. If the operand does not evaluate to a positive integral value, zero is used at the count limit.

The ‘ADVANCE’ keyword parameter is optional and has meaning only when the source file is transmitted by the `/*EXECSQL` interface. If present, ‘ADVANCE’ indicates that the DB2 result set row cursor should be advanced to the next row each time the group is repeated.

<%leave%> Statement Syntax

The HTML extension statement ‘`<%leave%>`’ forces an immediate exit from an iterative group, regardless of the current repetition count.

Each ‘leave’ statement consists of:

- The research keyword ‘leave’ (in upper or lowercase).
- An optional label name.

Use the following format to code this statement:

```
<%LEAVE label%>
```

‘Leave’ can only be specified within a ‘do’ group, and is invalid if used outside of a delimited group.

If a ‘leave’ is encountered without a label, it causes processing of the current, inner-most ‘do’ group to be completed. A label can be used to specify the completion of a ‘do’ group at an outer, higher nesting level.

<%next%> Statement Syntax

The HTML extension statement ‘`<%next%>`’ causes a logical skip to the top of an iterative group.

Each ‘next’ statement consists of:

- The research keyword ‘next’ (in upper or lowercase).
- An optional label name.

Use the following format to code the statement:

```
<%NEXT label%>
```

‘Next’ can only be specified within a ‘do’ group, and is invalid if used outside of a delimited group.

If a ‘next’ is encountered without a label, it causes a logical skip to the top of the current, inner-most ‘do’ group to be completed. A label can be used to specify a skip to the top of a ‘do’ group at an outer, higher nesting level.

<%enddo%> Statement Syntax

The HTML extension statement ‘`<%enddo%>`’ indicates the end of a ‘do’ group, and must be present for the group to be valid.

Each ‘enddo’ statement consists of:

- The reserved keyword, ‘enddo’ (in either upper or lowercase),
- An optional label name, which must match the name coded for the preceding ‘do’ statement.

Use the following format to code this statement:

```
<%enddo label%>
```

‘Enddo’ delimits the end of the last group initiated by the last unterminated ‘do’ statement. If the label name is used, it must match the label name specified on the ‘do’ statement.

Operation Of Iterative Groups

During evaluation processing, when each ‘do’ statement is first encountered, the HTML extension facility resets that group’s HTXINDEX value to zero and then evaluates the repetition count operand. If the evaluation:

- Yields a positive integral value, that value is used as the repetition limit for the group.
- Does not yield an integral value, zero is assigned as the limit and evaluation proceeds with the source line following the terminating ‘enddo’ statement.

During the first iteration of the group, the index value is set to one. The cursor advance option for the group is ignored. At each subsequent iteration, the current index value (HTXINDEX) is compared to the repetition limit value.

- If the index value, plus one, exceeds the repetition limit value, the group is terminated and evaluation proceeds with the source line following the ‘enddo’ statement.
- If the index value, plus one, is less than the repetition limit value, the index value is incremented by one. If the ‘ADVANCE’ option was coded, the DB2 result set cursor is advanced to the next row. This action occurs only when HTML extension processing is invoked by the `/*EXECSQL` interface, otherwise, it is ignored.

A ‘leave’ statement encountered during evaluation of the group causes immediate termination of the group.

A ‘next’ statement encountered during evaluation of the group causes evaluation to skip to the top of the group.

Iterative Group Example

The following illustrates the use of an iterative group and the HTXINDEX variable, including nested references to HTXINDEX and use of a label name.

```
<HTML>
<BODY>
<p>
<%if www.field.0 EQ 0%>
  No query variables were sent with this URL.
  </BODY>
  </HTML>
  <%exit%>
<%else%>
The values of query variables sent with this URL are:
<%enddo%>

<%loop: do www.field.0%>
  <P>Variable <%htxindex.loop%> Name =
  '<%www.field.<%htxindex.loop%>.name%>'.
   <P>Variable <%htxindex.loop%> Value =
  '<%www.field.<%htxindex.loop%>.value%>'.
<%enddo loop%>

</BODY>
</HTML>
```

The next example illustrates how iteration is used in conjunction with the `/*EXECSQL` process section to display DB2 result rows. “Name”, “job”, and “salary” are DB2 column names.

```
<HTML>
<BODY>
<p>
Salaries of all non-management personnel:
<TABLE BORDER>
<THEAD>
<TH>Name
<TH>Salary
</THEAD>
<TBODY>
<%do EXECSQL.ROWS ADVANCE%>
  <%if JOB EQ 'MGR'%>
     <%next%>
  <%endif%>
  <TR>
  <TD><%name%>
  <TD><%salary%>
<%enddo%>
</TBODY>
</TABLE>
</BODY>
</HTML>
```

Other HTML Extension Statements

The following extention statements are available:

<%date(_)%> Statement Syntax

The HTML extension '`<%date(_)%>`' statement causes the processor to insert the date at this point in the output stream. The date can be generated in the following formats:

Format	Description	Output
date(e)	European	DD/MM/YY
date(j)	Julian	YYDDD
date(o)	Ordered	YY/MM/DD
date(s)	Standard	YYYY/MM/DD
date(n)	Normal	DD MMM YYYY
date(u)	USA	MM/DD/YY
date(h)	HTTP	DAY, DD MMM YYYY HH:MM:SS GMT

Table 7–2. The <%data(_)% Statement Syntax

If an invalid date option is specified or if date() is specified, date(n) is assumed. Date(h) generates an HTTP formatted date according to current specifications. If the HTTP specification changes, the output of this selection also changes.

<%time(_)%> Statement Syntax

The HTML extension '`<%time(_)%>`' statement causes the processor to insert the time at this point in the output stream. The time can be generated in the following formats:

Format	Description	Output
time(c)	Civilian	HH:MMxx where xx = AM or PM.
time(l)	Long	HH:MM:SS.nnnnnn
time(n)	Normal	HH:MM:SS

Table 7–3. The <%time(_)% Statement Syntax

If an invalid time option is specified or time() is specified, time(n) is assumed.

<%exit%> Statement Syntax

The HTML extension '`<%exit%>`' statement causes the processor to immediately terminate expansion of the current source file. It is equivalent to encountering the end of the input source file.

Each '`<%exit%>`' statement consists of the reserved keyword, 'exit' (in either upper or lowercase). Code this statement as follows:

```
<%exit%>
```

DB2 Result Set Cursor Advance

When output file tailoring is invoked by `/*EXECSQL`, only the first result set row from the DB2 query is available unless you force the internal row cursor to position on subsequent rows.

There are two types of cursor advance:

- Auto advance at each iteration of a code the '`<%do%>`' statement by coding an 'ADVANCE' keyword. This forces the row cursor to advance to the next row each time the iteration group is restarted.
- Explicitly controlled cursor advance using the `<%advance%>` statement.

HTTP Response Control Statement

An HTML extension "response" statement is used to customize the HTTP protocol response headers transmitted when a dynamic page is output. The only response function supported in the this release is addheader. It allows you to create customized HTTP response header output.

<%Response.AddHeader%> Statement Syntax

The HTML extension '`<%response.addheadert%>`' statement causes the processor to invoke the `SWSRESP("add")` function to buffer a customized HTTP protocol response header.

Each '`<%response.addheader%>`' statement consists of:

- The reserved keyword, 'response.addheader' (in either upper or lowercase).
- A string literal giving the HTTP response header name.
- The operand value of the HTTP response header.

Code this statement in the following form:

```
<%RESPONSE.ADDHEADER "name" "value"%>
```

<%Response.AddHeader%> Usage

Use this extension statement to indicate how a particular page is to be handled by the browser. Some possible uses are:

- Specify a back-dated “`Expires:`” response header or “`Pragma: no-cache`” to ensure the Web browser does not re-display a page from it's cached copy by coding the following:

```
<%response.addheader "Expires" "Mon, 01 Jan 1997 12:01:00
GMT"%><%response.addheader "Pragma" "no-cache"%>
```

- Force the browser to set a cookie value if the current page is ever retrieved by coding:

```
<%response.addheader "Set-cookie" "SeenThisPage=YES"%>
```

- Force the page to be displayed in a new browser window by coding:

```
<%response.addheader "Window-target" "_blank"%>
```

HTML Extension Merge Processing

The HTML extension Facility has special merge processing interfaces to other Shadow OS/390 Web Server turn-key components. With merge processing you can insert the results of special queries directly into an HTML page.

*Interface With /*EXECSQL*

When a `/*EXECSQL` process section is executed, it can automatically invoke file output processing for:

- Generation and output of a input form, when the INPUTFORM keyword is specified.
- Tailoring of the actual SQL statement before execution.
- Generation and output of an output page, when the OUTPUTFORMAT keyword is specified.

Special statement types and variable values are made available by the HTML extension facility when processing the output of a DB2 query. These statements and variables are available for use only when the source file, specified by an OUTPUTFORMAT keyword of an `/*EXECSQL` process section, is expanded.

*Special /*EXECSQL Variables*

When a DB2 result set is being formatted by the HTML extension facility, the following special variables are available for customizing source file output:

EXECSQL Variables	Description
EXECSQL.ROWS	Contains the number of result set rows returned by the execution of the SQL statement It is provided primarily for use as the iteration limit operand of a '`<%do%>`' statement, but it can be used anywhere.
EXECSQL.ROW	Contains the current result set row number, beginning with row 1.
EXECSQL.SQLSTMT	Contains the fully expanded text of the SQL statement actually executed.
EXECSQL.PARTIAL	Used to indicate if a partial result set was returned for a query. • **`NO`** = The result set was not truncated. • **`YES`** = The result set was truncated based on the limitations specified by MAXROWS or MAXBLOCKS parameters of EXECSQL.
EXECSQL.COLUMNS or EXECSQL.COLUMN.0	Are synonymous. They evaluate to an integer count of the number of result set columns returned by the DB2 query.
EXECSQL.COLUMN.n	There is one EXECSQL.COLUMN.n variable built for each column of the result set. The evaluated value of this operand is the column name. By requesting double evaluation of this symbol, you cause the underlying column data to be referenced: • `<%EXECSQL.COLUMN.3%>` is evaluated as the name of the third result set column. • `<%<%EXECSQL.COLUMN.3%>%>` is evaluated as the contents of the third result set column within the current result set row.
Result Set Column Names	Each result set column, within the current result set row, can be referenced directly using the name of the column. For the SQL statement, "`select name, job, salary from Q.staff`", reference the variables, "`name`", "`job`" and "`salary`" within a source file as HTML extension statement operands. When a result set column has no pre-assigned name, such as when the SQL statement, "`select avg(salary) from q.staff`" is executed, the server assigns column names in the form "`COLn`' where '`n`' is the column number relative to one. In this example, the name would be "`COL1`". If a pre-assigned column name has embedded blanks, Shadow OS/390 Web Server replaces these with underscore characters.

Table 7–4. Special /*EXECSQL Variables

CHAPTER 8: Automated State Management Facility (ASMF)

This chapter discusses the Automated State Management Facility, what it does, why it's important, and how it works. If you need more information on protocols, refer to Chapter 1, "An Overview," or see the *Shadow OS/390 Gettting Started Guide*.

What is a Stateless Protocol?

HTTP is a stateless protocol, which means there are no inherent mechanisms in the protocol that relate any transaction to those that precede or follow it. This stateless quality exists partly because no permanent communications session exists following each individual request/response interaction, such as the sessions used for 3270 terminal communications. When a long-lived session exists, both client and server implicitly retain an active 'bind' to their session partner. As long as the bind exists, the partners do not change; this allows the execution sequence to be tightly controlled.

Persistent Session Support

Because HTTP allows 'Persistent Session Support', it is sometimes misinterpreted as a means to remember application-dependant state information. HTTP's 'Persistent Session Support' only allows an existing communication pathway to be reused without tearing-down and re-building overhead. The HTTP protocol explicitly limits 'Persistent Session Support' to only this type of path reuse, because there is absolutely:

- No guarantee that the session will be reused.
- No requirement about the sequence in which additional requests and responses are transmitted over the open session.
- No assumption that the pathway is actually connected with the original partner.

What is ASMF?

When you create complex web-based applications, you usually need a series of client/server interactions to remain logically related to ensure proper operation. ASMF implements many basic interfaces that are necessary to maintain in-flight application "state" information across multiple HTTP request/response interactions. These include:

- The Server-side token API
- Persistent GLOBAL. variable support

- A means to dynamically insert application-generated data values into outbound HTML pages
- Easy access to inbound query variables and HTTP cookies
- The ability to control 'Set-Cookie:' and other HTTP response headers

Why use ASMF?

ASMF encapsulates the details of these basic Server interfaces into a unified, powerful, and simple-to-use interface. By automatically handling HTTP protocol-related issues and other details involved when using the basic Server interfaces, developers can devote their full attention to application requirements.

What Constitutes 'State' Information?

'State' information is any piece of information that must be 'remembered' at the completion of a client/server interaction, because it has some bearing on a future, anticipated interaction. It could be:

- A single data value, such as an account number that is keyed only once by the end user, but remembered and used across many subsequent client/server query interactions.
- A very complex collection of 'state' elements which 'remember' user choices, control user or application generated 'holds' on resources, or even determine future processing sequences.

Whatever information is 1) known by any single client/server interaction and 2) must also be remembered or re-generated in order for completion of subsequent interactions, constitutes 'state'.

Transmitting State Information

State information must temporarily or permanently reside at either the client or the server between HTTP transactions. HTTP client's almost always bear responsibility for saving and later communicating some fragment of the state information to the server as part of the inbound request, while the HTTP servers almost always bear the responsibility for setting everything up for the exchange. It is only the server that has any interest-in or knowledge-of the information needed for subsequent state transaction processing.

The Transmission Process

Step 1: Client sends a single request to the server.

The client is only allowed to transmit a single request and to receive a single response; no additional handshaking is possible between the client and server within the confines of an individual interaction. This means the client must predictively transmit all the information needed on the inbound request.

Step 2: Server accesses and interprets the request.

If enough information is received and it is valid, the server processes the request. If there is not enough information or the information is invalid, the server does a rescan and returns an error message.

Step 3: Server prepares the information to transmit.

The server must predictively generate directions for the client that instruct the client in creating its next request. This means that between requests, application-dependent information must be stored somewhere. This information could:

- (Often the best choice) Reside wholly on the client in the form of a cookie or an HTML hidden form variable.
- (Recommended if security an issue) Be divided between both the client and the server.
- Reside entirely on the server. (This is often impossible to do, but if possible, it is not recommended.)

  Usually, HTTP servers are not able to unilaterally save and then restore application-level state information because most IP addresses are 'leased' temporarily from a network controller. The same gateway IP address could be shared consecutively by many downstream clients, because each unique IP address can only be related to a specific client for the duration of a single transaction. (This may not be true for some, very tightly controlled, hard-wired Intranet configurations, but such a configuration is extremely rare!)

Step 4: The server transmits information back to the client.

HTTP clients (typically web browsers) are able to save state information as either HTTP cookies or HTML hidden form variables.

HTTP cookies: The server must transmit the cookie to the client, and specify under what future conditions each cookie should be transmitted with inbound requests. The HTTP clients must be configured to accept cookies for this to work.

HTML forms: The server must transmit appropriately coded HTML pages to the client. The client saves the information as form values and then transmits these values to the server as query variables. If the server does not transmit these values correctly, the client cannot generate them. It is the responsibility of the application to code the HTML pages as the server does not scan or modify the HTML.

Step 5: Client attempts to receive information by using one of the following:

HTTP cookies: If cookies are used, the HTTP clients may or may not be configured to handle or store them. The cookies could be discarded and the state information would not be available to send the next request.

HTML Pages: If the information is not received in the appropriate HTML format, the client will be unable to generate the necessary query variable for its next inbound request.

Step 6: The Client attempts to send another request.

The client attempts to send another request to the server.

- **Sends state information.** The client transmits the state information to the server either as a query variable or an inbound HTTP cookie.
- **Does not send state information.** If, for any reason, the client does not transmit any state information, then it is unlikely that the server will be able to continue with the series of related transactions. The server needs a guide for retrieving or reconstructing server-side values, and other client-supplied information upon which to base continuing state operations. The lack of any state information from a client constitutes a null or 'starting' state that can be usefully interpreted by the server application.

Step 7: These steps are repeated.

The process of sending and receiving information continues until either:

- The cycle is complete and all transactions in the series have been generated and returned to the client.
- An error occurs.

Server-Side State Information

Due to security-related concerns or other application considerations, you can keep most of the state information out of direct view of the client by storing it at the server. The server will always require some value from the client (if only a null set) in order to determine which set of saved information to use.

There are two methods of retaining state information on the server.

Method 1: Using the Server-side Token Facility

Shadow OS/390 Web Server's server-side token facility provides:

- A generic means for retaining server-side state information.
- A means for disposing of the information after a preset timeout period.

State information can be stored within a server-token, and associated with a unique 24-byte token key. Only the 24-byte token key is transmitted to the client. During subsequent interactions, and upon receipt of a valid token key from the client, the original state information can be retrieved at the server. The server can also detect if the token data and key have become invalid because the inactivity timeout has passed or because the server has been restarted.

Method 2: Using ASMF

ASMF options automate the use of the Web Server's basic token service for retaining state information at the server and handling the token key lookup process when the key is transmitted from the client. ASMF handles token-based service requests using either HTTP cookies or HTML form variables.

The current version of Shadow OS/390 Web Server does not provide persistence for basic server-side tokens. After a product restart, token keys become invalid, because the tokens, their keys, and associated data are lost when the product is shutdown. If you need to use persistent server-side state information, you must save and restore the data from within your application. Although you will not be able to use the built-in token processing options, ASMF can readily be used for handling application "key and data" values via HTTP cookies.

Although the Web Server's generalized token facility is used internally by the ASMF, do not attempt to use both interfaces simultaneously. Data values stored by the ASMF facility should not be retrieved using direct calls upon the token access API routines, and the token access API routines should never be used to update the data values held by the ASMF facility.

Using HTTP Cookies

HTTP cookies, originally implemented in Netscape browsers, have become the state information transport of choice for most applications during the last two and a half years. They:

- Allow more robust methods for saving and transmitting state information than do HTML form/query variables.
- Provide a means for disposing of stale information.
- Can be made persistent for long periods of time.
- Can be limited in scope so that they apply only to specific URL requests.
- Can be sent by the browser whenever it accesses your host domain and the specific or general URL path within it. (Your application-generated HTML or hyperlinks do not need to be preset in the brower’s window for this to occur.)

Remember

When designing an application:

- Individual users can block the use of HTTP cookies. This is usually done out of fear that personal or sensitive information could be wrongly used. (Such fears are only partially rooted in fact. See CIAC Bulletin I-034 of March 1998 for a discussion of these security-related issues.)
- Some browsers discard HTTP cookies under a variety of circumstances.

Application designers should take these issues into account and weight the benefits of HTTP cookies against the possible problems, which can occur when they are not re-transmitted as anticipated.

Using HTML Forms

Client browser software allows state information to be saved as HTML form variables during 'think time', and it even provides the means to transmit these values as query variables on a subsequent interactions. To mask the state information from direct view, use the 'type=hidden' form variables. This does not, however, prevent the end user from examining the information when viewing the HTML source text unless the information has been enciphered.

The ASMF can be used to automate storage and retrieval handling of application state values using server-tokens that have been placed in hidden HTML form variables. It is your application's responsibility to ensure the token key is properly embedded into the HTML forms transmitted to the client. The server does not scan or modify the HTML.

Using State Information Sets

ASMF supports three variations of information sets: COOKIE, CTOKEN, and FTOKEN.

COOKIE	This uses HTTP cookies as a communications transport. The actual encoded value data, not a token ID, is sent to and from the client.
CTOKEN	This uses HTTP cookies as a communications transport. A server-side token ID is sent to and from the client. The value data is stored within the server-side token, and is accessed indirectly using the token ID.
FTOKEN	This uses URL query variables as a communications transport. A server-side token ID is sent to and from the client. The value data is stored within the server-side token, and is accessed indirectly using the token ID.

Each type defines how state information is transmitted to and from the client, and whether server-side tokens are used to indirectly access information held at the server.

Characteristics	COOKIE	CTOKEN	FTOKEN
HTTP Cookies are used for transmission of state information.	X	X	
Query variables are used to transmit state information from the client to the server.			X
Value data and keys are transmitted to and from the client in encoded form.	X		
Value data and keys are stored in a server-side tokens. Only the server token ID is transmitted to and from the client.		X	X
The value data and keys remain valid so long as they are retained and retransmitted by the client.	X		
The server token expires after a preset timeout period after which the value data and keys can no longer be accessed.		X	X
The server automatically generates the HTTP "Set-cookie:" response header which is sent to the client when the information set is created.	X	X	
The server automatically re-creates the information set when it receives a specially formatted HTTP "Cookie:" request header.	X	X	
The application, not the server, is responsible for predictively setting up the appropriate query variable information which the client will send on each inbound request.			X

State Information Set Name

Each State Information Set must have a unique, application-assigned name. State information names:

- Must be from 1 to 16 bytes in length.
- Must start with one of the U.S. English alphabetic characters, A through Z.
- Must use the following characters in the remainder of the name:
  - U.S. English alphabetic characters, A through Z
  - The digits, 0 through 9
  - The underscore (EBCDIC x'6D') character
- Can be any combination of upper and lower case letters, but it is always processed internally as an uppercase string.

State Information Set Variables

Each State Information Set conceptually consists of a collection of like-named variables that the server specifically manages. Each variable in the collection has a name which is, or begins with, the two-level qualifier:

```
GLVSTATE.setname
```

where "setname" is the application-assigned name of the State Information Set.

Applications use the server's built-in variable access functions to read or write these special collection variables. The server, while working in the background, intercepts all references to these variables and automates many of the chores required for maintaining the transaction state.

A web transaction program can:

- Create, delete, or re-initialize an information set by assigning a pre-defined value to the collection's Control Variable.
- Determine the validity of the state information by interrogating the collection's Status Variables.
- Specify options for handling the state information set by setting the collection's Option Variables.
- Save or retrieve application-dependent data values by setting or interrogating the collection's Value Variables.
- Refer to state information sets or keyed value data indirectly using Indexed Access Variables.

GLVSTATE. Variable Inventory

The following table lists all valid GLVSTATE. variable name symbol forms along with an indication of each one's general usage and applicability to each type of information set.

Variable Name	Variable Type:	Applies To:
GLVSTATE.setname	Control & Status Variable	Any Set
GLVSTATE.setname.STATUS	Status Variable, Read-Only	Any Set
GLVSTATE.setname.TYPE	Status Variable, Read-Only	Any Known Set
GLVSTATE.setname.TOKENID	Status Variable, Read-Only	Token-Based Sets
GLVSTATE.setname.TIMEOUT	Option Variable	Token-based Sets
GLVSTATE.setname.DOMAIN	Option Variable	Cookie-based Sets
GLVSTATE.setname.PATH	Option Variable	Cookie-based Sets

Variable Name	Variable Type:	Applies To:
GLVSTATE.setname.EXPIRES	Option Variable	Cookie-Based Sets
GLVSTATE.setname.SECURE	Option Variable	Cookie-Based Sets
GLVSTATE.setname.VALUE	Value Data	Any Known Set
GLVSTATE.setname.VALUE.keyname	Value Data	Any Know Set
GLVSTATE.setname.KEY.0	Indexed Access, Read-Only	Any Known Set
GLVSTATE.setname.KEY.keyindex	Indexed Access, Read-Only	Any Known Set
GLVSTATE.setname.KEY.keyindex.VALUE	Indexed Access	Any Known Set
GLVSTATE.0	Indexed Access, Read-Only	All Known Sets
GLVSTATE.setindex	Indexed Access, Read-Only	All Known Sets
GLVSTATE.setindex.<suffix>	Indexed Access	All Known Sets
GLVSTATE.0.TRACE	Facility Control	All Known Sets
GLVSTATE.0.DELETE	Facility Control	All Known Sets

The following values in the above table are used to denote user-selectable names or index numbers values.

Value	Description
setname	User selected name of the State Information Set.
keyname	User selected name of a value data key that is only used with keyed value sets. Valid value key names are from 1-to-48 bytes in length and must be composed of the same characters used for valid set names (such as, begin with alphabetic character, contain only alphanumeric characters plus underscore).
keyindex	Positive integer corresponding to the Nth value data key (used only with keyed value sets).
setindex	Positive integer corresponding to the Nth State Information Set known to the application.
<suffix>	Indicates that any trailing name suffix(s) is valid for use with a non-indexed GLVSTATE.setname. Reference can also be used in this position for indirect indexed access.

GLVSTATE. variable symbol cannot exceed 84 bytes in length; this includes all qualifiers. It is impossible to use both the maximum size setname and the maximum size keyname for most, otherwise valid, symbol name combinations.

Collection Control Variable Name

An information set's control variable is the primary means of controlling the set or determining whether a set exists. The control variable name is:

```
GLVSTATE.setname
```

where "setname" is the application-assigned name of the State Information Set.

Evaluation of GLVSTATE.setname

When you interrogate the value of an information set's control variable, one of the following two status values is returned. (These are the only two values that are ever returned, even if the query is made by a REXX-language procedure against a previously unknown/un-initialized variable.)

NULL	A NULL (zero-length) string is returned if the value information within this set is un-usable. For example, if the set does not exist ('`UNKNOWN`' status), or if it is associated with an expired server-side token ('`EXPIRED`' status), this value will be returned as an overall indication of the non-usability of the set.
'VALID'	This string is returned only if the State Information within the set is usable by the application. For example, it is returned for sets that have either an '`ACTIVE`' or '`RESTORED`' status.

Assigning Values to GLVSTATE.setname

By assigning a pre-defined value to a set's control variable, the underlying state information set is created, deleted, or re-initialized. The following values can be assigned:

Pre-defined Value	Description
'NEW(type,timeout)'	Used to create the set. If a set with this name already exists, the existing set will be destroyed and a new one will be created with the same name. Only the string, 'NEW', is required; the parentheses, type, and timeout parameters are optional. If omitted, the server assumes the value 'NEW(COOKIE)'. The type parameter indicates the type of State Information Set to be created; it must be a COOKIE, CTOKEN, or FTOKEN. The timeout parameter: • Is only valid with CTOKEN and FTOKEN. It is not valid with COOKIE and must be omitted with 'NEW(COOKIE)'. • Must be coded as an unsigned integer that specifies the time limit (in seconds) to be applied to the server-side token associated with the information set. The server-side token is destroyed if it is not used within that time limit. If it is not coded, the default time limit for all server tokens is used.

Pre-defined Value	Description
'DELETE'	Used to destroy the State Information Set. If there is a server-side token associated with the set, it is also destroyed at the same time. If the set is currently unknown, no action occurs. Deleting an information set at the host does not stop the information from being retransmitted from the client. The server does not invalidate/expire HTTP cookies or HTML form variables stored at the client.
'RESET'	Reset is used to change the status of an existing information set. • **'ACTIVE' Status** - Resetting has no effect upon the information set. • **'RESTORED' Status** - Resetting places the set in 'ACTIVE' status. For cookie-based information sets, the cookie is re-transmitted in the next outbound response, but there is no substantive effect for FTOKEN-type sets. The domain name, path, expiration date and secure parameters used for HTTP cookie processing are not re-constructed; they must be set separately, if needed. • **'EXPIRED' Status** - Resetting places the set in 'ACTIVE' status. A new associated token ID and token are assigned. (The old token ID is not re-used). This action does not re-create the expired value data and for HTTP cookie-based sets does not re-construct the domain, path, expiration date and secure parameter information.

Collection Status Variables

Status variables allow you to determine the exact status and type of a set. These variables are read-only and derive their evaluated values from internal state information set processing activities. The status variables are:

```
GLVSTATE.setname.STATUS
GLVSTATE.setname.TYPE
GLVSTATE.setname.TOKENID
```

where "setname" is the application-assigned name of the state information set.

Evaluation of GLVSTATE.setname.STATUS

When you interrogate the value of this variable, one of the following values is returned. (These are the only values ever returned, even if the query is made by a REXX-language procedure against a previously unknown/un-initialized variable.)

Variable Value	Description
'UNKNOWN'	This value is returned if the corresponding set does not exist and is currently unknown to the server.
'ACTIVE'	This string is returned by one of the following reasons: • The state information was created during operation of the current transaction. • A 'RESTORED' set was reset. For cookie-based sets, HTTP 'Set-cookie:' response headers are generated automatically by the server when outbound transmission buffers are flushed to the client.
'RESTORED'	This string is returned for any state information set that has been successfully re-created by the server using inbound query or cookie information. All value data, originally saved in the set when it was created, is usable by the application.
'EXPIRED'	This string is only returned for token-based information sets. Inbound query or cookie variables processed by the server contain a server-side token ID. Because the token has expired, all value data, originally saved in the set when it was created, is no longer available.

Evaluation of GLVSTATE.setname.TYPE

When you interrogate the value of this variable, a COOKIE, CTOKEN, OR FTOKEN values is returned.

Note:
These values are only returned for information sets that are known to the server. For unknown sets, a null string is returned to high-level-language callers. REXX-language callers will receive a string containing the name of the variable, itself, corresponding to the REXX-language specification for handling of un-initialized variables.

Evaluation of GLVSTATE.setname.TOKENID

When you interrogate the value of this variable:

- **The server-side token ID value (CTOKEN and FTOKEN).** Information sets are returned and displayed as a 24-byte, hexadecimal value, unique from all other server-assigned token ID values.
- **COOKIE-based and for unknown sets.** A null string is returned to high-level-language callers. REXX-language callers will receive a string containing the name of the variable corresponding to the REXX-language specification for handling of un-initialized variables.

Using COOKIE-Type Information Sets

HTTP cookies are:

- Used as the communications transport for application state information and application information in encoded form.
- Transmitted directly to and from the client.

How Cookies Work

1. Create a cookie-type state information set.
2. Save the application data into it as either unkeyed or keyed data values.
3. Run the application.
4. Once application ends, the server generates a 'Set-Cookie:' response header for every cookie-based information set your application has created. (This is a normal end-of-transaction transmission process.)

   The 'Set-Cookie:' response consists of a name / value pair. The name is derived by prefixing the state information set name you assigned with the specially-recognizable string, 'SWSSTATE_'. The value is a representation of your application's data, in encoded form.
5. The Web browser re-transmits this name / value pair back to the server in an HTTP 'Cookie:' request header.
6. As part of request parsing, the server checks all in-bound 'Cookie:' headers for the special string prefix, 'SWSSTATE_'. If found, the server re-constructs the original set of information saved by your application.

Benefits of Using Cookies

HTTP cookies are the most versatile transport mechanism, because:

- The browser can send them whenever it accesses your host domain and the specific or general URL path within it. (Your application-generated HTML or hyperlinks do not need to be preset into the browser's window for this to occur.)
- HTTP cookies can be made persistent. The browsers can continue to send the cookies, even when it contacts your site infrequently.

Some Considerations

This type of information set is ideal for saving user profile information or other non-sensitive data. Because your application's data is actually transmitted to and from the browser, you:

- Must ensure the data's validity, since it can easily be spoofed or altered.
- Should avoid storing sensitive value data at the browser unless using SSL communications sessions to/from a browser that is in a secured location.

Using CTOKENS and FTOKENS

Use CTOKEN-type or FTOKEN-type information sets instead of COOKIES, if you:

- Need a short-term, server-controlled time limitation.
- Want to retain sensitive data only at the server.

Creating a Cookie-type Set

The following shows how to use Shadow/REXX statements to create a COOKIE-type information set named, "EMPDATA", and save profile data into the set.

```
GLVSTATE.EMPDATA                  = 'NEW(COOKIE)'      /* Create Cookie Set */
GLVSTATE.EMPDATA.VALUE.NAME     = 'John F Fields'
GLVSTATE.EMPDATA.VALUE.TITLE    = 'Product Author'
GLVSTATE.EMPDATA.VALUE.PRODUCT = 'Shadow Web Server'


GLVSTATE.EMPDATA                = 'NEW(COOKIE)'    /* Create or  re-initialize */
GLVSTATE.EMPDATA.VALUE.NAME     = 'John F Fields'
GLVSTATE.EMPDATA.VALUE.TITLE    = 'Product Author'
GLVSTATE.EMPDATA.VALUE.PRODUCT = 'Shadow Web Server'
```

The unOfficial HTTP Cookie Specification

HTTP cookies, originally invented by Netscape to maintain persistent client state information, were implemented in their 2.0 release. Although these specifications were never officially adopted by the Internet's standards organizations, they have been widely adopted by all modern Web browsers and Web servers. Attempts to define an official standard for HTTP Cookies have not been successful, thus Netscape's specification remains the de facto standard. (See Cookie I-D Drafts for a list of the many revisions to the "official" RFC2109 HTTP Cookie Specification.)

To open a separate window describing Netscape's original HTTP Cookie specification, click this hyperlink: http://home.netscape.com/newsref/std/cookie_spec.html. We strongly suggest you familiarize yourself with this specification if you intend to use HTTP Cookies as a transport mechanism for application state information.

Comments and suggestions for the use of HTTP Cookies in Shadow OS/390 Web Server refer to the Netscape specification.

Anomalies You May Encounter

Although the Automated State Management Facility (ASMF) provided by Shadow OS/390 Web Server normally hides all the details of working with HTTP cookies, your application must be designed to appropriately use them. Failure to take into account how Web browsers support and use HTTP cookies could result in some or all of the following anomalies:

- No cookies are returned by the Web browser, as anticipated, for any application's URLs.
- No cookies are returned by the Web browser, as anticipated, for some application's URLs, but not for others. The inbound cookies are transmitted or omitted in what seems to be a random pattern.
- Duplicately named cookies are sent by the Web browser for certain application's URLs, but not for others -- again, in a seemingly random fashion.

Making HTTP Cookies Work Reliably

Normally, anomalies arise because of MVS-centric thinking when authoring both static HTML pages and dynamic applications. If you remember that the Internet was originally devised and operated solely on UNIX-based platforms, and then act accordingly, these problems are less likely to occur.

Keep the following points in mind:

- Most Internet URLs, like the UNIX file system, are normally formed as a hierarchal directory and sub-directory path, with a filename specification at the end. Although Shadow OS/390 Web Server allows you to map almost any

URL string to a WWW rule definition, if your application deviates widely from these assumed UNIX norms, you could experience problems using HTTP cookies.

- Most UNIX file systems process names in a case-sensitive fashion. A file (and by implication a URL) name, such as "/MYFILES/ABC.HTM", does not designate either the same path or file as "/myfile/abc.htm", or "/MyFile/Abc.htm".
- Leave nothing to chance. Most users will type a URL location into a browser window in order to access the first page of an application, but few will overtype a URL which has been linked-to within the application. The first page of your application should specify explicit values for the host domain and URL in all hyperlinks and HTML forms that route control back to your application.

Some Suggestions

Here are some suggestions that should help ensure that HTTP Cookies operate reliably in conjunction with your application.

Firewalls

When running inside a firewall:

- Give Shadow OS/390 Web Server a domain name during start-up, even if it is not an officially registered name.
- Make sure that the domain name you assign looks like an officially registered domain name with at least 3 qualifiers separated by periods (2 qualifiers if the name ends with .com, .edu, .net, .org, .gov, .mil, or .int.

If you cannot assign a character-format domain name, then use the IP dot-notation address instead.

Why?

Often a test MVS system (inside a firewall) will not have an official domain name assigned. For HTTP, these systems are normally addressed using IP dot-notation addresses. However, a problem can occur if your more expert in-house users have populated PC-resident "HOSTS" or "LMHOSTS" files with short, character-format names. When these names are used as a short-cut for specifying the longer IP dot-notation addresses, particularly if the short names are also specified at the server, problems will develop.

For example, if the short name "MVSA" has been assigned to one of your in-house MVS systems as a shortcut for a longer IP address, such as 12.77.88.43.

Because these short, single-qualifier domain names do not fully conform to the Netscape specification for domain names, HTTP cookies are sometimes not transmitted as anticipated. Such a variance can be further complicated, not only by the server's host domain name parameter setting, but by how the end-user originally typed-in the domain name at the browser. This is often many transactions before any application has even had a chance to generate an HTTP "Set-cookie:" response.

Setting up a standard domain name for the server and then using the server provided SWSINFO function to retrieve and insert it into outbound responses:

- Guards against the original, hand-typed domain name being used as a default by the browser throughout a series of URL interactions.
- Ensures a conforming value will always be generated by the server itself.

Front-end (gateway) transactions and HTML pages generating outbound HTTP cookies should be coded to explicitly specify the domain name in URLs used for HTML form ACTION= parameters or HTML hyperlinks. This explicit domain name will override the original URL domain name typed by the user.

Of course, these steps and warnings do not apply when your server has an officially assigned Internet domain name.

Use only Lowercase Letters

If you place a domain name reference into an HTML document as a hyperlink or as part of a form's ACTION= parameter, or within an HTTP response header, use only lowercase letters.

Why?

Microsoft I/E Web browsers appear to store host domain names internally as lowercase values.

We have observed instances in which HTTP cookies were not transmitted back to a host when a document hyperlink explicitly specified the domain name using uppercase letters.

For this reason, the server automatically converts the host domain name start-up values to lowercase strings.

Coding URL Hyperlinks

Choose either upper or lowercase letters when coding all URL hyperlinks and then use them consistently. Although Shadow OS/390 Web Server processes these values in a caseless manner, some Web browsers do not.

Why?

Some Web browsers process URL path values in a case-sensitive manner. If you set up an HTTP cookie for an uppercase URL such as "/MYPATH/xyz.htm", the browser may not send it back if an embedded hyperlink specifies the URL in lowercase, such as, "/mypath/xyz.htm".

Path, Not Prefix

Where ever the original Netscape cookie specification uses the word, "path", it literally means "path", not "prefix".

Why?

Browsers transmit HTTP cookies with any inbound request that has a "path" matching the "path" explicitly specified, or implied, when the cookie was originally received from the server.

The path specification, "/abc", will match URL requests such as "/abc/file1.htm" or "/abc/prog02", but not URL requests, such as, "/abcfile1.htm" or "/abcprog02".

CHAPTER 9: Executing User Programs

Shadow OS/390 Web Server has a built-in facility for executing user written transaction programs for processing URLs. To implement this facility define a `/*PROGRAM` process section within a WWW rule instead of a REXX (or other) process section.

Program Process Sections

Each `/*PROGRAM` process section header statement names the user program executed, how the program is invoked, whether the user program is pre-loaded, and specifies the parameter values passed to the program. Other parameters can be specified on a `/*PROGRAM` header statement.

What Programs Can be Executed

You can execute user written Web transaction programs written in COBOL, C, or PL/I. These programs must be compiled and linkage edited into a load library accessible to Shadow OS/390 Web Server. User written transaction programs use the High-Level Language (HLL) interface API (Application Program Interface) to access data values sent inbound with the URL request and to transmit results back to the Web browser.

If you need facilities that are unavailable within Shadow/REXX, you can execute other REXX language interpreters. This is accomplished using the `/*PROGRAM` process section interface. See “Using Other REXX Interpreters” on page 9-4.

- Consult the *Shadow Programming Guide* or the online HTML files for the API Index of all REXX-Language and High-Level Language (HLL) interface routines.
- Information specific to each of the three supported High-Level Languages can be found on the following pages:
    - “Writing COBOL Web Transaction Programs” on page 9-6
    - “Writing C/370 Web Transaction Programs” on page 9-6
    - “Writing PL/I Web Transaction Programs” on page 9-7

Where is CGI?

Shadow OS/390 Web Server does not provide a Common Gateway compliant Interface (CGI) for user written transaction programs. Instead, a more efficient programming interface is provided for thread based architecture using Shadow/REXX.

Standalone command procedures and programs can be executed within an Auxiliary TSO/E server address space. This environment provides the same essential characteristics of the CGI interface, but has the advantage of being a commonly used MVS transaction processing environment.

A future release of Shadow OS/390 Web Server will provide additional CGI compliance using these Auxiliary TSO/E servers. See Using TSO/E Services for Web Transaction Processing for more information.

Where the Programs Must Reside

Programs to be executed as Web transaction processors must be named within a `/*PROGRAM` process section of a WWW rule. Program load modules must reside in one of the following locations:

1. A user program library, allocated to DD name `'SWSRPCLB'` during Web server subsystem start-up
2. The product's load library, allocated as STEPLIB
3. The MVS system's linklist libraries

The server always attempts to locate user programs within the libraries in the order shown above. If programs are to be loaded from `'SWSRPCLB'`, the library must be allocated at the time the subsystem is started; it cannot be dynamically allocated once the server has been started.

Note:

We strongly recommend that user written High-Level Language programs be loaded from the `'SWSRPCLB'` dataset. Doing so avoids possible naming conflicts with load modules that comprise the Shadow OS/390 Web Server product.

The product's load library (allocated as STEPLIB) must be APF-authorized. Updates to an APF-authorized library in most customer sites requires special administrative procedures and authorization. However, the user program library, `'SWSRPCLB'`, does not inherently require APF-authorization.

Coding PROGRAM Process Sections

To define a PROGRAM process section, code the WWW event procedure definition as follows:

```
/*WWW  /aURLvalue   ... WWW header stmt keywords ...
/*PROGRAM         ... program header stmt keywords ...
```

The following keywords can appear on the `/*PROGRAM` statement:

Keyword	Description
TYPE( )	(Required) Valid Operand Values: • **MODULE** = This is the only supported operand value for this release. It indicates that the NAME keyword specifies the name of an MVS load module to be executed.
NAME( )	(Required) Valid Operand Values: Specify the name of the MVS load module to be executed to process this Web transaction.
INVOKE( )	(Required) Valid Operand Values: • **CALL** = Specifies that the load module is invoked using a BASR instruction. • **LINK** = Specifies that the load module is invoked using the MVS LINK SVC.
PRELOAD( )	(Required) Valid Operand Values: • **YES** = Specifies that the load module is pre-loaded into storage when that WWW rule is enabled. • **NO** = Specifies that the module is loaded into storage each time it is used.
HEADERS( )	(Optional) Valid Operand Values: • **YES** (default) = Specifies that a simple HTTP response header is generated automatically by the system. The HTPP response header includes only the standard status "`code (200)`", the "`Server: response element`", and specifies "`Content-type: text/html`". • **NO** = Specifies that the module is completely responsible for generating the outbound HTTP response header.
PARM( )	(Optional) Valid Operand Values: A 1 to 100 byte string. The value coded for this keyword is passed to the load module using standard MVS linkage conventions (register 1 point to a full word containing the address of the two-byte parameter string length followed by the string).

Table 9–1. Program Statement Header Keywords

Keyword	Description
SUBSYS( )	(Optional) Valid Operand Values: Specifies the name of the DB2 subsystem needed for the current application. The connection is completed before control is passed to the application program. This eliminates the need for the application program to use DSNALI to set up its own DB2 thread. You can specify "?" in place of an actual 4-byte DB2 subsystem name. When SUBSYS(?) is coded, the Server substitutes the value of the DEFAULTDB2SUBSYS start-up parameter in place of "?". The DB2 subsystem must be active when the program is started. If the DB2 subsystem is not active, the program is not executed. When any SUBSYS keyword operand is coded on a /*PROGRAM statement, it is required that the PLAN keyword also be explicitly coded. Note: If your program does not use DB2, you should not code either the SUBSYS or PLAN keywords, or as an alternative, code SUBSYS(NONE) and PLAN(NONE).
PLAN( )	(Optional) Valid Operand Values: Specifies the name of the DB2 plan which is used to establish a connection to DB2. The connection is completed before control is passed to the application program. This eliminates the need for the application program to use DSNALI to set up its own DB2 thread. You can specify "?" in place of the actual 8-byte DB2 plan name. When PLAN(?) is coded, the Server substitutes the value of the DEFAULTDB2PLAN start-up parameter in place of "?". The DB2 subsystem must be active when the program is started. If the DB2 subsystem is not active, the program is not executed. When the PLAN keyword is used, the SUBSYS keyword must also be specified.

Table 9–1. Program Statement Header Keywords

Using Other REXX Interpreters

Shadow OS/390 Web Server supports the use of Shadow/REXX as well as other REXX interpreters for coding REXX-language process sections directly within WWW rules definition. You might need one of these other interpreters if you have special addressing environments which are supported only from TSO/E or CA-OPS/MVS REXX.

Executing a Non-Shadow/REXX Interpreter

To execute a non-Shadow/REXX interpreter name, enter the REXX interpreter as the user program module in the `/*PROGRAM` process section. Because almost all batch-mode REXX interpreters take the name of the procedure to be interpreted as a parameter, you pass the procedure name using the PARM operand of the `/*PROGRAM` header statement.

For example, in order to execute an IBM TSO/E REXX routine named `INFOSYS`, request that the IBM interpreter load module, `IRXJCL`, be executed. The REXX procedure name, `INFOSYS`, is passed as the parameter value:

Example

```
/*PROGRAM TYPE(MODULE) -
NAME(IRXJCL) -
INVOKE(LINK) -
PARM(INFOSYS other parameter values)
```

Or, to execute a CA-OPS/MVS REXX procedure, name the OPS/MVS interpreter, OI, as the program module. This passes the REXX procedure name as a parameter.

Note:
NEON Systems ships two sample members within the NEON WWW ruleset which illustrates how to invoke a CA-OPS/MVS procedure as a Web transaction procedure. The supplied procedure performs an MVS "D A,L" operator command and returns the results to the Web browser.

To enable and run this sample, you must have Computer Associate's CA-OPS/MVS installed on your system. Comments within the OPSMVS member of the NEON WWW ruleset describe steps you must take to enable and run this sample application. The supplied URL value is "`/NEON/OPSMVS/DAL`".

Run-time Environment of Other REXX Interpreters

Other REXX interpreters are executed as a Web transaction processing subtask within Shadow OS/390 Web Server's address space. If you code procedures using TSO/E (or other) REXX interpreters, remember that Shadow OS/390 Web Server's address space is not a TMP (Terminal Monitor Program). The run-time environment within the server's address space supports only those functions which are available for batch-mode execution of the interpreter.

Most commands of the "ADDRESS TSO" host command environment are NOT AVAILABLE regardless if you are executing TSO/E REXX or Shadow/REXX. Do not count on:

- The use of ADDRESS TSO within TSO/E REXX to support dynamic allocation functions (the ALLOC and FREE TSO commands).
- Any other TSO/E command processor.
- Using any ISPF facilities which do not operate in batch execution mode.

Web Server APIs for Other REXX Interpreters

Refer to the *Shadow Programming Guide* or to the online HTML to see which REXX built-in functions and which language codes are available by non-Shadow/ REXX interpreters. When used with other REXX interpreters, some restrictions or operational differences can apply to the built-in functions. Even if you are familiar with using the functions from Shadow/REXX, consult the documentation for any differences before assuming that the operation is exactly the same.

Writing C/370 Web Transaction Programs

To compile and successfully execute Web transaction programs written in C and compiled with IBM's C/370 compiler, observe the following restrictions and conventions:

- Include the following pragma at the top of all main routines:

  ```
  #pragma runopts(noexecops,nospie,nostae,plist(mvs))
  ```

  This `pragma` applies to V2 R1 version of the IBM C/370 compiler. The options specified could be different for other releases of the compiler.

- Include the API definition header file, `'sccphd.h'`, supplied within Shadow OS/390 Web Server's routine. This file contains definitions for constants and external entry points used to invoke Web server API services.

  ```
  #include "sccphd.h"
  ```

- The header file contains typedef statements which define Web server API parameter argument types.

- The member, `'SWCW1'`, in the NEON-supplied sample library, contains a functional C/370 example that you can compile and link. It illustrates the required structure of C/370 Web transaction programs and some simple Web server API invocations.

Writing COBOL Web Transaction Programs

To compile and successfully execute Web transaction programs written in COBOL, observe the following conventions and restrictions:

- For COBOL II, specify the following compile time options:

  ```
  S,XREF,NORES,NODYNAM,NORENT,MAP,LIST,APOST,TRUNC(BIN)
  ```

  And these options for the linkage editor:

  ```
  XREF,LET,LIST,MAP,NORENT,AMODE(31)
  ```

- For COBOL/370, specify the following compile time options:

  ```
  S,XREF,LIB,MAP,LIST,APOST,RENT,TRUNC(BIN)
  ```

And these options for the linkage editor:

```
RETN,AMODE=31,RMODE=ANY
```

- Copy the API definition file, `'SBCPHD'`, supplied with Shadow OS/390 Web Server. This file contains definitions for constants and external entry points used to invoke the server's API services.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY SBCPHD.
```

- The supplied `'SBCPHD'` member contains 77-level constants relating API function names (such as SWSSEND, SWSVALUE, or SWSTOKEN) to their actual load module entry point names (such as SWCPSN, SWCPVL, SWCPTK). These constants are supplied for reference purposes only.

  When calling Web server API routines from COBOL, you must specify the actual load module entry point name (such as, SWCPSN). Failure to do so can result in compiler generated dynamic loading of subroutines. This is not allowed or supported within Shadow OS/390 Web Server's address space.

  For instance, to invoke the SWSSEND function, code the API invocation as:

```
CALL 'SWCPSN' USING  ... parameters ...
```

- The member, `'SWCBLW1'`, in the NEON-supplied sample library contains a functional COBOL example that you can compile and link. It illustrates the required structure of COBOL Web transaction programs and some simple Web server API invocations.

Writing PL/I Web Transaction Programs

To compile and successfully execute Web transaction programs written in PL/I, observe the following conventions and restrictions:

- Declare the main entry point of your PL/I program as follows:

```
WEBPGM: PROC(PARMSTRING) OPTIONS(MAIN,REENTRANT);
DCL PARMSTRING CHAR(100) VARYING;
```

  This example illustrates the required OPTIONS values as well as shows that a standard MVS parameter list can be passed in varying character format.

- Copy the API definition file, `'SPCPHD'`, supplied with Shadow OS/390 Web Server. This file contains definitions for constants and external entry points used to invoke Web server's API services.

```
%INCLUDE SPCPHD;
```

- Make the following declaration within your main routine to suppress certain PL/I run-time options.

```
DCL PLIXOPT CHAR(100) VARYING INIT('NOSPIE,NOSTAE')
STATIC EXTERNAL;
```

- Do not use FETCH or RELEASE statements if SSL is used for encryption of session data. This is due to the language environment restrictions for VM and MVS.

The member, `'SWPLIW1'`, in the NEON-supplied sample library contains a functional PL/I example which you can compile and link. It illustrates the required structure of PL/I Web transaction programs and some simple Web server API invocations.

CHAPTER 10: Writing DB2-Based Web Applications

You can create DB2-based Web transactions using Shadow OS/390 Web Server in many different ways. You can:

- Write a High-Level Language application program which executes SQL statements and returns the output to the Web client.
- Write a REXX-language WWW procedure which uses the facilities of Shadow REXXTOOLS to execute SQL statements.
- Create transaction definitions using an `/*EXECSQL` process section.

Shadow/REXXTOOLs DB2/SQL Interface

The documentation on Shadow/REXXTOOLs tools is in the HTML online files under the topic heading, Writing DB2-Based Web Applications.

/*EXECSQL Process Sections

Shadow OS/390 Web Server has a built-in facility for executing host-based DB2 operations. This facility is implemented by defining an EXECSQL process section within a WWW rule instead of a REXX (or other) process section.

Each EXECSQL process section header statement:

- Names the DB2 subsystem and plan to use.
- Gives the SQL statement to execute.
- Specifies how the results of the DB2 operation should display.
- Specifies the input HTML form used to collect query statement parameter values.

EXECSQL process sections provide a simple, yet extremely powerful, means of porting complex DB2 queries to the World Wide Web.

Operation of EXECSQL Sections

When a URL matches a WWW rule containing an EXECSQL section, a transaction procedure is initiated which performs the following actions:

- (optional) Transmits an input HTML form to the end user to collect input values which are substituted as parameter values into the SQL statement.
- Opens a thread to the requested DB2 subsystem and plan.

- Specifies which SQL statement in the EXECSQL section is executed. Parameter values can be used to control the size of a query result set.
- Processes the returned result set to create an end user display. Options allow a DB2 result set to be formatted automatically as an HTML table or DB2 column data that can be inserted into an output form.

The EXECSQL process section intrinsically handles the:

- Display of input forms.
- Connection to DB2.
- Execution of an SQL statement.
- Formatting of a result set.

Most DB2 queries can be constructed using EXECSQL in a matter of minutes.

Coding EXECSQL Process Sections

To define a EXECSQL process section, code the WWW event procedure definition as follows:

```
/*WWW  /aURLvalue   ... WWW header stmt keywords
/*EXECSQL           ... EXECSQL header stmt keywords
  SQL statement to be executed
```

In this example, the `/*EXECSQL` definition consists of three elements:

- The WWW header statement
- The EXECSQL process section header statement
- An SQL statement to be executed

EXECSQL Header Statement Keyword

Each keywords which can appear on the `/*EXECSQL` statement are listed in the following table. Some keywords enclose nested keyword values.

Keyword	Description
`SUBSYS( )`	Specify the DB2 subsystem targeted for the SQL operation being executed. If the SUBSYS keyword is not coded, the start-up parameter, DEFAULTDB2SUBSYS, is used to specify the DB2 subsystem name.
PLAN( )	Specify the name of the plan used for execution of the SQL statement. If the PLAN keyword is not coded, the start-up parameter, DEFAULTDB2PLAN, is used to specify the plan name. Any DB2 plan name can be specified. However, the plan must contain the NEON-supplied DBRM, `'SWRXSQ'`.

Table 10–1. Valid /*EXECSQL Parameters

Keyword	Description
MAXROWS( )	Specify an integer to limit the number of rows returned for the query. The transaction is aborted if more than the specified number of rows are accessed. If this operand is not specified, there is no limit on the number of individual rows returned.
MAXBLOCKS( )	Specify an integer to limit the number of 50K blocks allocated to contain the result set. If the block limit is exceeded, the SQL statement is aborted. If this operand is not specified, the start-up parameter, PREFETCH, is used as the limit of 50K blocks.
INPUTFORM( )	Specify this keyword to transmit an HTML entry form to the client. In order for this to work, no query variables can accompany the inbound URL request. If they do, the SQL statement is executed and the result set is formatted and transmitted. If this operand is not specified, no entry form is transmitted to the client. The following keywords are coded as sub-operands of the INPUTFORMAT keyword. One of the sub-operands DDNAME and DSNAME must be specified. • **`DDNAME( )`** Specifies the DD name of the dataset which contains the HTML form to be transmitted. This sub-operand cannot be specified if the DSNAME sub-operand is specified. • **`DSNAME( )`** Specifies the dataset name which contains the HTML form to be transmitted. This sub-operand cannot be specified if the DDNAME sub-operand is specified. • **`MEMBER( )`** Specifies the PDS member name which contains the HTML form to be transmitted. This sub-operand is required if the dataset is a PDS.
AUTOFORMAT( )	Specify this keyword to automatically format the SQL result set as an HTML table. Use either AUTOFORMAT or OUTPUTFORMAT as an EXECSQL keyword. (If OUTPUTFORMAT is coded, do not specify AUTOFORMAT.) The following keywords are coded as sub-operands of the AUTOFORMAT keyword. • **`TITLE( )`** (Required) The 1 to 64 byte string literal used as the title for the automatically formatted HTML page. The operand can be enclosed in quotes. • **`TABLE( )`** (Optional) Specifies the 1 to 64 byte string literal to be inserted into the <TABLE> tag generated by the automatic format operation. If omitted, the value 'BORDER' is assumed. • **`BODY( )`** (Optional) Specifies the 1 to 64 byte string literal to be inserted into the <BODY> tag generated by the automatic format operation. If omitted, a null string is assumed. • **`IGNORE( )`** Code up to 8, 30-byte column names to ignore during automatic formatting. Ignored columns are not formatted as part of the HTML result set table. If any names contain embedded blanks, enclose each column name in quotation marks.

Table 10–1. Valid /*EXECSQL Parameters

Keyword	Description
OUTPUTFORMAT( )	Specify this keyword to format the SQL result set by invoking HTML Extension processing; special merge processing interfaces are provided for this facility. Use either AUTOFORMAT or OUTPUTFORMAT as an `/*EXECSQL` keyword. (If AUTOFORMAT is coded, do not specify OUTPUTFORMAT.) The sub-operands of this keyword specify the text file used as a template for creating the resultant client display. The following keywords are coded as sub-operands of the OUTPUTFORMAT keyword. One of the sub-operands DDNAME and DSNAME must be specified. • **DDNAME( )** Specify the 1 to 8 byte DD name of the dataset containing the output template. This sub-operand cannot be specified if the DSNAME is specified. • **DSNAME( )** Specifies the dataset name containing the output template. This sub-operand can not be specified if the DDNAME is specified. • **MEMBER( )** Specifies the 1 to 8 byte PDS member name of the dataset containing the output template. This sub-operand is required if the dataset is a PDS. • **CONTENTTYPE( )** Specifies the 1 to 50 byte string literal used as the MIME Content-type: specification. If this operand is omitted, ‘text/html’ is assumed.
ERRORURL	Specify this keyword to use an alternate Auxiliary Error URL to process the error. This replaces the Neon supplied URL of `SYSTEM/ERROR/AUX`.

Table 10–1. Valid /*EXECSQL Parameters

SQL Statement

The following apply to SQL statements:

- Code the SQL statement to be executed after the `/*EXECSQL` header statement.
- The SQL statement can continue across multiple input lines.

To Parameterize the SQL Statement Using Variables

- Code the HTML extension insertion statements wherever user specified parameter values are entered.

To Continue a String Literal across an Input Line Boundary

- End the continued line with a plus sign (+). This signals continuation of a literal and forces the next line to be concatenated immediately behind the preceding line (after removing the plus sign).

EXECSQL Examples

Sample One

The following example is one of the sample transactions supplied with the Shadow OS/390 Web Server as URL "/NEON/SQLEXEC2".

- When the transaction is first accessed, no query variables are sent inbound with the request. The EXECSQL processor transmits the input form, instead of processing the SQL statement.
- The end user completes the form and presses the submit button. The form runs the same /*EXECSQL rule using the same URL value, except this time, query variables are sent inbound.
- Because query variables are sent with the request, the EXECSQL processor executes the SQL statement. The result set data is automatically formatted into an HTML table and sent back to the end user.

Sample One's /*EXECSQL rule contains the following:

```
/*WWW  /NEON/SQLEXEC2
/*EXECSQL  MAXROWS(400) -
    INPUTFORM( DDNAME(HTMFILE) -
            MEMBER(SAMPSQL2) -
           ) -
    AUTOFORMAT( TITLE('Search Results') -
            BODY('bgcolor="#FFCC33"') -
           )
    Select * from Q.Staff where job = '<%www.var.jobname%>'
                  order by <%www.var.order%>
```

Sample Two

In this sample transaction, the URL, "/NEON/SQLEXEC3", is processed in exactly the same way as the first one, with an input form and automatic output formatting. However, in this example, the input form solicits the user to key an entire SQL statement, which is then executed.

If an invalid SQL statement is entered,/*EXECSQL routes controls to a special server error procedure, "SYSTEM/ERROR/AUX". This procedure sends an HTML page specially formatted to present abnormal conditions encountered when processing /*EXECSQL and other built-in server turn-key facilities. It formats more complex reason information than does "SYSTEM/ERROR/500".

Sample Two's /*EXECSQL rule contains the following:

```
/*WWW  /NEON/SQLEXEC3
/*EXECSQL  MAXROWS(400) -
    INPUTFORM( DDNAME(HTMFILE) -
            MEMBER(SAMPSQL3) -
           ) -
    AUTOFORMAT( TITLE('Search Results') -
            BODY('bgcolor="#FFCC33"') -
           )
    <%www.var.sqlstmt%>
```

Sample Three

This sample transaction if for the URL "`/NEON/SQLEXEC4`". It performs approximately the same function as the first sample; however, this sample uses the `OUTPUTFORMAT` parameter to specify a file in which the DB2 result data is inserted. Shadow OS/390 Web Server HTML Extension Statements are used to format the result set data in a customized Web page rather than as a standard HTML table.

Sample Three's `/*EXECSQL` rule contains the following:

```
/*WWW  /NEON/SQLEXEC4
/*EXECSQL  MAXROWS(400) -
    OUTPUTFORMAT( DDNAME(HTMFILE) -
               MEMBER(SQLEXEC4) -
                  CONTENTTYPE(text/html) -
                 ) -
    select * from q.staff
```

Notes Regarding Samples

These sample WWW transactions were installed with Shadow OS/390 Web Server; however, they may not be operational on your system due one of the following reasons:

- The samples were deleted after the product was installed.
- A different DB2 subsystem name or plan name needs to be specified for these WWW rules to execute properly.
- The sample DB2 table, `Q.STAFF`, is not present on your system.

CHAPTER 11: Using TSO/E Services For Web Transaction Processing

You can create Web transactions which use the services of TSO/E to perform a procedure and build a client response. The TSO/E transaction processing environment is supported by Shadow OS/390 Web Server through the auxiliary TSO/E address spaces.

How TSO/E Auxiliary Servers Operate

There are three basic functions to operating the TSO/E auxiliary server:

Starting When the auxiliary server facility is activated, Shadow OS/390 Web Server issues the MVS **START** command(s) to initialize one or more auxiliary TSO/E servers. The server always initializes the minimum number of servers specified by the TSOMINSERVERS startup parameter.

Monitoring

During operation, the server can start additional auxiliary server address spaces, but it is limited by the maximum designated by the TSOMAXSERVERS startup parameter. An additional auxiliary server starts each time the “waiting for execution” command queue reaches a threshold limit. The threshold limit is specified by the TSOSRVQUEUEADD parameter.

Periodically during operation, Shadow OS/390 Web Server checks to see if more auxiliary servers are started than the minimum server value. If yes, and if any server has been dormant for a specified period of time, the auxiliary server is terminated. (The dormancy time value is set by the TSOSRVDORMANTTIME startup parameter.) This action keeps the number of active auxiliary server address spaces in line with the actual work load being processed by the server.

Stopping When the Shadow OS/390 Web Server is shutdown, it automatically terminates all auxiliary TSO/E servers.

Auxiliary Server Operation

Although each auxiliary TSO/E server is initialized as an MVS started task, each operates as a batch mode Terminal Monitor Program (TMP). During batch mode operations, TSO/E:

- Reads each command to be processed from the `'SYSTSIN'` dataset.
- Writes all responses to the `'SYSTSPRT'` dataset.

Direct terminal output (such as TGET and TPUT) cannot be used during batch mode operation since there is no terminal session active.

JCL

The JCL used to start each auxiliary server contains a `'SYSTSIN'` and `'SYSTSPRT'` DD statement which specifies the `SUBSYS=` keyword instead of an actual dataset location. The `SUBSYS=` keyword points to Shadow OS/390 Web Server's MVS subsystem identifier. Only TSO/E address spaces that have been initialized by Shadow OS/390 Web Server are allowed to connect to it via the `SUBSYS=` DD statement.

When the TMP (`IKJEFT01`) opens the `'SYSTSIN'` and `'SYSTSPRT'` datasets, these datasets are intercepted by Shadow OS/390 Web Server. The main Shadow OS/390 Web Server's address space is then able to send a command to the TSO/E server by writing the command through the `'SYSTSIN'` pipe. It intercepts the output generated by each command by reading it from the `'SYSTSPRT'` pipe.

Using TSO/E Commands in Web Transactions

TSO/E commands can be passed to an auxiliary address space (via `'SYSTSIN'`) for execution as part of a Web transaction definition. The transaction waits for the command to execute within the auxiliary server. When the command completes, the original transaction reads the response (from `'SYSTSPRT'`) and creates a response for the Web client.

Time Limits

When Web transactions are submitted with a TSO/E command for execution, the server implements a time limit. If that limit is exceeded, the original Web transaction is signaled that a time-out has occurred and it is not allowed to continue operation. This facility keeps Web transactions from waiting for an indefinitely long time, when a response is not forthcoming.

User IDs

Each time a new command passes through the `'SYSTSIN'` pipe, the run-time effective userid of the requesting Web transaction is propagated into the TSO/E address space. This means any command operating within an auxiliary address space is authorized to access the same MVS system resources that the original Web transaction is allowed to access.

Restrictions on Commands Executed Within an Auxiliary Server

Commands processed within an auxiliary server must conform to the constraints of batch mode TSO/E operation. The commands cannot:

- Attempt to read or write to an actual terminal session.

- Attempt to read additional data from `'SYSTSIN'`. Only the command, along with additional parameters, can be passed to the TMP as a command request. If a command procedure issues a **READ** or **GET** request to `'SYSTSIN'`, it is cancelled by the server.

Resource Restrictions

Shadow OS/390 Web Server imposes limitations on the resources which any single command can consume. Each command scheduled into an auxiliary server is cancelled if:

- The original Web transaction's time limit is exceeded.
- It exceeds a fixed limitation on wall clock time allowed for execution of any single command.
- It enters a sustained wait state which exceeds a fixed limitation on wall clock time allowed for waits by any single command.
- It exceeds a fixed limitation on CPU time allowed for execution of any single command.
- It generates excessive output which exceeds a fixed limitation on the amount of output allowed per command.

If any of these limitations are exceeded, the server cancels the TSO/E command procedure and re-dispatches the original Web transaction.

When a command is cancelled for one of the above reasons, the entire auxiliary server is normally cancelled and restarted. This action is part of the timekeeping function and is designed to ensure a “clean” operating environment for any TSO/E commands subsequently scheduled.

Release Restrictions

This release of Shadow OS/390 Web Server does not support the use of Web server Application Programming Interfaces from within an auxiliary TSO/E address space. A future release of Shadow OS/390 Web Server will provide cross-memory support for many of these API interfaces.

Activating the TSO Server Facility

Shadow OS/390 Web Server contains an optional facility which allows you to create Web transactions using the capabilities of TSO/E. Because the server does not operate as a Terminal Monitor Program (TMP), TSO/E services are not directly available within the server's address space. However, Shadow OS/390 Web Server still allows you to use the services of TSO/E to create transactions.

The Web server does this by starting, monitoring and controlling up to 30 additional TSO address spaces. TSO/E commands can be scheduled into one of these out-board TSO tasks for execution. The output for each command is then intercepted and used to create Web transaction responses.

To activate this optional facility, you must:

- Set up the out-board TSO server JCL.
- Configure Shadow OS/390 Web Server's start-up parameters.

Setting Up the SWSTSO Started-Task JCL

The SWSTSO member of the installation CNTL library contains the JCL procedure needed to run each auxiliary TSO server address space. Shadow OS/390 Web Server's main address space issues MVS **START** commands to bring up the out-board TSO servers.

You must tailor the JCL (found in the `'SWSTSO'` member) as follows:

1. Change the LOADLIB parameter to contain the name of Shadow OS/390 Web Server's load library or add your own load libraries to the STEPLIB concatenation defined within the JCL.

2. Change the REXXLIB parameter to contain the name of Shadow OS/390 Web Server's SYSEXEC library or add your own EXEC libraries to the SYSEXEC concatenation defined within the JCL.

3. Tailor or add any other dataset definitions required to execute the TSO within the out-board address space.

4. Copy the updated `'SWSTSO'` member into a procedure library which is searched by the MVS **START** command (this could be SYS1.PROCLIB).

If you are running a security product (like RACF, ACF/2, or Top Secret), you might need to define a userid for the out-board TSO server address spaces. To do this, associate the name with the SWSTSO start-up procedure, and then set up access rules so the userid can access required datasets.

Configuring Initialization Parameters

You must set the values of various start-up parameter to configure the out-board server facility. These are specified by coding additional statements within the SWSxIN00 start-up parameterization EXEC procedure.

Example:

```
"MODIFY PARM NAME(TSOSRVACTIVE) VALUE(YES)"
```

The parameters, which control the auxiliary server facility, are explained in the following table.

Parameter Name	Description
TSOSRVACTIVE	• **YES** - Enables the out-board TSO server facility. • **NO** - (default) Disables the out-board TSO server facility.
TSOSRVTIMELIMIT	Specify a default time limit, in hundredths of a second, that a WWW rule waits for a TSO command to complete. If the command takes longer, the WWW rule revokes the command request.
TSOMINSERVERS	Specify an integer value between 1 and 30 for the minimum number of out-board TSO servers that the SWSS main task starts and monitors.
TSOMAXSERVERS	Specify an integer value between 1 and 30 for the maximum number of out-board TSO servers that the SWSS main task starts and monitors.
TSOSRVMAXQUEUE	Specify an integer value giving the number of wait-for-execution queue entries that the server manages.
TSOSRVQUEUEADD	Specify a threshold value. Whenever the wait-for-execution queue reaches this number of active entries, the server starts a new TSO server address within the TSOMINSERVERS and TSOMAXSERVERS limitations.
TSOSRVDORMANTTIME	Specify the time, in seconds, after which a dormant server is stopped, when more than TSOMINSERVERS are running.
TSOSRVCMDRUNTIME	Specify the maximum amount of wall clock time, in seconds, any single TSO command is allowed to run. If the limit is exceeded, the TSO command process (and server) is canceled with a `S322 abend` (and then restarted automatically, if the MIN/MAX values allows it).
TSOSRVWAITTIME	Specify the maximum amount of wall clock time, in seconds, any single TSO command is allowed to place itself into a wait for state. If the limit is exceeded, the command is canceled.
TSOSRVCPUTIME	Specify the maximum amount of CPU time (in seconds) that any single TSO command can use before being canceled with an `S322 abend`.
TSOSRVMAXLINES	Specify the maximum number of PUT operations any single command can issue to `'SYSTSPRT'`. The server cancels itself with `S722` if this limit is exceeded.
TSOSRVPROCNAME	Specify the name of the started procedure that SWS uses to start each server. The default value is `'SWSTSO'`.
TSOSRVSWAPPABLE	• **YES.** Makes each TSO server address space swappable. • **NO.** Makes each TSO server address space non-swappable.
TSOSRVSTARTUPPARM	Use this parameter to pass additional start-up parameters to the `'SWSTSO'` procedure. The string specified here is appended to the **START** command issued by SWSS

Table 11–1. Initialization Parameters for the Out-Board Server

Building Shadow/REXX Based TSO/E Web Transactions

As discussed in the Shadow/REXX documentation (see the online HTML documentation), the services of TSO/E are not directly available to Web transaction procedures written in Shadow/REXX. For example, the following Web transaction procedure fails, because TSO/E is not available within the Shadow OS/390 Web Server's main address space.

```
/*WWW /ThisFails
   /*REXX
     ADDRESS TSO
     "LISTA"
```

The LISTA command fails to execute because there is no TMP (Terminal Monitor Program) available to process the command. (Only the EXECIO is supported in this environment and only because NEON Systems has implemented EXECIO directly within Shadow/REXX.)

To get around this limitation, Shadow OS/390 Web Server implements the ADDRESS TSOSRV host command environment. ADDRESS TSOSRV differs from the more familiar ADDRESS TSO interface, because it schedules execution of the TSO/E command within an out-board address space where the TMP (Terminal Monitor Program) is available.

ADDRESS TSOSRV - Command Types

When a command is passed to the ADDRESS TSOSRV host environment, it falls into one of two command classes:

Pseudo-Commands

Pseudo-commands are processed directly by the ADDRESS TSOSRV interface, and are used to parameterize the interface. For instance, a pseudo-command can be used to specify the amount of time the interface waits for subsequent TSO/E commands to complete execution.

Any valid TSO/E Command

Any command string that is not a pseudo-command passes to an out-board TSO server for execution. The results of these commands (the output generated to the `'SYSTSPRT'` dataset) are placed on the REXX external data queue. The original command requestor can retrieve this output, reformat it, as needed, and generate a browser response.

Pseudo Command Formats

The following table lists the pseudo-commands that can be passed to the ADDRESS TSOSRV interface.

Command Name	Operand	Command Action
SETTIMEOUT	An integer between 1 and 90,000	Sets the time limit, in hundredths of a second, that the interface waits for execution of a TSO/E command in an out-board server.
GETTIMEOUT	None	Returns the current execution timeout value in a message on the external data queue. For example: `TSO REMOTE EXECUTION TIMEOUT VALUE SET TO n` where n gives the current timeout value.
SETOUTPUTTRACE	YES or NO	Turns `'SYSTSPRT'` tracing on or off within the interface. When output tracing is on, each message that the TSO/E command issues to `'SYSTSPRT'` is logged in the wrap-around trace
STRIPPROMPTS	YES or NO	Specifies handling of messages issued by the TMP. These messages include the echoing of the TSO/E command string, certain RACF or ACF/2 status messages, and the `READY` prompt when each command completes.

Table 11–2. Pseudo Commands Used with ADDRESS TSOSRV

Issuing a Pseudo-Command

Issue a pseudo-command as follows:

```
ADDRESS TSOSRV         /* issue following to TSOSRV environment */
"settimeout 200"       /* wait 2 seconds for command completion */
```

Return Values

If the operand of a pseudo-command is invalid, the interface sets the built-in RC (return code) variable to 8. For successful execution of a pseudo-command, the RC variable is set to the value 0.

TSO/E Command Formats

Command Formats

Except for pseudo-commands, any command string passed to the ADDRESS TSOSRV interface is scheduled for execution within an out-board TSO server address space. The command is executed, and the results are placed into the external data queue of the REXX procedure that issued the ADDRESS TSOSRV request.

Run-Time Userid

The run-time effective userid, under which the current Web transaction procedure operates, is propagated to the TSO server when the command is scheduled for execution. This means the authorization level of the command executed in the out-board server cannot surpass the authorization of the original Web transaction subtask that scheduled it.

TSO/E Commands for Out-board Execution

Almost any valid TSO/E command can be scheduled for out-board execution. This includes simple built-in TSO/E commands, such as "**LISTALC**", the name of a REXX or CLIST procedure, or invocations of ISPF/PDF or other TSO/E command processors. The command name and arguments pass as a single string via the ADDRESS TSOSRV interface.

The maximum size of each TSO/E command (and it's arguments) is limited by the interface to 320 bytes in length.

Return Values

The ADDRESS TSOSRV interface sets a return code value into the built-in RC variable for each command issued. You need to examine this variable to determine if the command was successful.

The following table lists the most common values returned in the RC variable following the execution request. If the execution fails and the code is not in this table, check the wrap-around trace or the MVS operator console. The server logs text messages describing the failure to one or both of these locations.

Value of `RC` Variable	**Meaning**
0 through 19	Execution was successful. If set to a non-zero value in this range, it indicates an exceptional condition resulted from the execution of the command.
Less than 0	Execution failed. The reason for the failure is logged in the server's wrap-around trace.
20	The out-board TSO server facility is not active.

Table 11–3. Return Values for ADDRESS TSOSRV

Value of RC Variable	Meaning
32	Execution of an out-board TSO command was rejected by the server's authorization facility.
48	Execution aborted. The command was scheduled for execution within an out-board server, but it did not complete within the allowed time-out period.
Other	The command could not be executed, or did not complete. The reason for the failure is logged in the wrap-around trace, or as a message issued to the MVS operator console from within the out-board server. Scheduling command requests within an auxiliary address space is a complex procedure. The interface can fail or cancel command execution for a number of reasons. For instance, out-board execution of a command can be cancelled due to excessive CPU time utilization (with an `S322 abend`), or invalid authorization to open a dataset (with `S913`).

Table 11–3. Return Values for ADDRESS TSOSRV

Queued Results

After successful execution of an out-board command, the results are saved to the external data queue of the requesting procedure. From there, they can be re-processed, if needed, by the requesting REXX procedure to generate an HTML page or other response. You can use the REXX-language `QUEUE()` function to interrogate the number of external data queue lines returned by the procedure, and the **PARSE PULL** instruction to retrieve each data line.

For large result sets, you may need to specify a large value for the QUEUESIZE parameter of the `/*WWW` rule header statement.

ADDRESS TSOSRV Example

The following example illustrates the use of ADDRESS TSOSRV in Shadow/REXX language WWW procedure. The command, “lista”, is executed within an auxiliary server address space and lists the allocations within the out-board server, not within Shadow OS/390 Web Server's main address space.

Note:
The sample `'SWSTSO'` started-task JCL supplied with the server allocates only a few datasets within each TSO server address space. Unless you modify this started-task procedure, or execute a command procedure within the server which allocates additional datasets, the **LISTA** command in the sample displays only 6-7 allocated datasets instead of the 100 or so allocated datasets you would normally see.

Example

```
/*WWW /listacmd  QUEUESIZE(400)
/*REXX
  ADDRESS TSOSRV              /* pass commands to TSOSRV envir. */
  "SETTIMEOUT 200"            /* wait 2 seconds for response    */

  "LISTA"                     /* issue LISTA in aux. server     */

  if RC <> 0 then do          /* handle bad return codes        */
    ADDRESS SWSSEND           /* switch environments            */
    "HTTP/1.0 500 Error"      /* generate HTTP response header  */
    "Content-type: text/html" /* specify type                   */
    ""                        /* required NULL line             */
    "<HTML><BODY>"            /* generate error response page   */
     "<H1>Error</H1>"         /* heading                        */
     "<P>Execution of the command"
     "failed with RC="RC      /* last part of message           */
     "</body></html>"         /* end of HTML page               */
     return                   /* exit this WWW procedure        */
  end                         /* end of error check             */

  ADDRESS SWSSEND             /* switch environments            */
  "HTTP/1.0 200 OK"           /* generate HTTP response header  */
  "Content-type: text/plain"  /* specify type                   */
  ""                          /* Required NULL line             */
  "The LISTA command returned the following:"
  " "                         /* insert blank line              */
  while queue() > 0           /* for each line returned by TSO  */
     parse pull dataline      /* get next SYSTSPRT line         */
     dataline                 /* write line to output stream    */
  end                         /* end of read loop               */
  return                      /* exit this WWW procedure        */
```

/*TSOSRV Process Sections

The Shadow OS/390 Web Server has a built-in facility for executing out-board TSO/E server command requests. This facility is implemented by defining a TSOSRV process section within a WWW rule. The TSOSRV process section is coded instead of a Shadow/REXX process section which contains an ADDRESS TSOSRV request.

Each TSOSRV process section header statement can specify the time limit placed upon external TSO/E command execution and the input HTML form to be used to collect query statement parameter values. TSOSRV process sections provide a simple, yet extremely powerful, means of building out-board TSO/E command procedure Web transactions.

Operation of TSOSRV Sections

When a URL matches a WWW rule containing an TSOSRV section, a transaction procedure is initiated, which performs the following actions:

- An input HTML form can optionally be transmitted to the end user to collect input values, which can be substituted as parameter values in the TSO/E command statement.
- The TSO/E command statement specified in the TSOSRV section is scheduled for execution within an out-board TSO/E auxiliary server address space. Parameter values can be used as command arguments on the statement.
- The TSO/E command output is intercepted by the Shadow OS/390 Web Server and transmitted directly to the end user.

The TSOSRV process section intrinsically handles the display of input forms, submission of the command to the external TSO/E server address space, and error recovery, if the command cannot be executed to completion.

Note:
The command procedure must directly create the out-board response, including the HTTP response header, HTML, or other data to be displayed.

Coding TSOSRV Process Sections

To define a TSOSRV process section, code the WWW event procedure definition as follows:

```
/*WWW  /aURLvalue   ... WWW header stmt keywords
   /*TSOSRV            ... TSOSRV header stmt keywords
     TSO/E Command statement to be executed
```

An TSOSRV definition consists of three elements:

- WWW header statement
- TSOSRV process section header statement
- Command statement to be executed in the auxiliary server

TSOSRV Header Statement Keyword

Each of the keywords which can appear on the `/*TSOSRV` statement are given in the table below. Some of these keywords enclose nested keyword values.

Keyword	Description
TIMELIMIT( )	Specify an integer in the range of 10 to 90000, giving the time limit imposed for command execution. The value specified is in hundredths of a second. This operand limits the amount of time the TSO/E command may take to complete. Execution of the command is canceled if the limit is exceeded.
INPUTFORM( )	Specify this keyword if you wish for the TSOSRV procedure to transmit an HTML entry form to the client. The entry form is only transmitted if no query variables accompany the inbound URL request. If query variables are present, the command statement is executed and the result set transmitted. If this operand is not specified, no entry form is transmitted to the client. The following keywords are coded as sub-operands of the INPUTFORMAT keyword. One of the sub-operands, either DDNAME or DSNAME, must be specified. • **`DDNAME( )`** Specifies the DD name of the dataset which contains the HTML form to be transmitted. This sub-operand can not be specified if the DSNAME sub-operand is specified. • **`DSNAME( )`** Specifies the dataset name with contains the HTML form to be transmitted. This sub-operand can not be specified if the DDNAME sub-operand is specified. • **`MEMBER( )`** Specifies the PDS member name that contains the HTML form to be transmitted. This sub-operand is required if the dataset is a PDS.

Table 11–4. TSOSRV Header Statement Keyword

TSO/E Command Statement

Code the command statement to be executed, following the `/*TSOSRV` header statement. The command statement may be continued across multiple input lines.

To parameterize the command statement, use variables by coding HTML extension insertion statements wherever user specified parameter values should be entered. To continue a string literal across an input line boundary, end the continued line with a plus-sign (+). The plus-sign signals continuation of a literal and forces the next line to be concatenated immediately behind the preceding line (after it removes the plus-sign).

TSOSRV Example

This example rule is handled by the server, as follows:

- When the transaction is first accessed, no query variables are sent inbound with the request. The TSOSRV processor transmits the input form, instead of processing the command statement line.
- The end user completes the form and presses the submit button. The form runs the same `/*TSOSRV` rule using the same URL value, except this time, query variables are sent inbound.
- Because query variables are sent with the request, the TSOSRV processor schedules the command statement for execution within an auxiliary server address space. The command procedure, `MYCLIST`, generates the resulting outbound response.

  The value of the run-time query variable `WWW.VAR.FILENAME` is substituted into the command statement before execution.
- If execution of the command fails for any reason, the `/*TSOSRV` processor regains control and transmits an error response page.

```
/*WWW  /NEON/SQLEXEC2
   /*TSOSRV   TIMELIMIT(400) -
              INPUTFORM( DDNAME(HTMFILE) -
                         MEMBER(SAMPTSO) )
   MYCLIST <%www.var.filename%>
```

Coding the External Command Procedure

The command procedure, executed within the auxiliary server, must create the entire response for the Web transaction. The output generated by the procedure (data written to `'SYSTSPRT'`), is sent directly to the Web client.

Data written to the `'SYSTSPRT'` dataset is interpreted by the server as follows:

- NULL data lines cannot be written to `'SYSTSPRT'`, and are discarded by the TMP (terminal monitor program). Do not attempt to write NULL data lines to `'SYSTSPRT'`, as the server is not able to intercept or process these.
- Any data line which beginning with a `X'00'` byte is interpreted by the server to be a BINARY data line. The server removes the leading `X'00'` byte and transmits the remainder of the data line unedited.
- If the data line begins with the five-byte sequence, `X'FF'||NULL`, the server interprets this as a request to transmit a NULL data line. The server transmits a `CF/LF` pair instead of the supplied data line.
- All other data sent to `'SYSTSPRT'` is interpreted by the server as `TEXT` output. Trailing blanks are removed, except the last one, if all of them are

blank. The data line is translated from EBCDIC to ASCII, and a CF/LF sequence is appended to the end of the line.

CHAPTER 12: AutoHTML - Web Enabling Transactions

In order to automatically generate Web transactions (AutoHTML), you must:

- Configure Shadow OS/390 Web Server to support the transaction server. See the *Installation Guide* for more information.
- Generate input and output transaction data format maps using Shadow OS/390 Web Server's Data Mapping Facility.
- Use the Data Mapping Facility to generate the HTML to display the output data on the Web browser.
- Build the rule to process the transaction.

IMS Implementation

Shadow OS/390 Web Server uses IMS/APPC in order to execute online IMS transactions and commands.

Installation and Configuration Overview

In order to use this interface, you must:

1. Install and configure the IMS Transaction Server to support IMS/APPC.
    - Configure IMS/APPC.
    - Install the NEON-supplied IMS LU 6.2 User Edit Exit (`DFSLUEE0`).
2. Configure MVS/APPC Support within Shadow OS/390 Web Server MVS/APPC Prerequisites.
3. Configure Shadow OS/390 Web Server for IMS transactions.
4. Enable IMS Transaction Server.
5. Verify the Installation.

Refer to the *Installation Guide* for detailed information.

Enable Web Transactions

Once you have the APPC interface configured, you must Web enable your IMS transactions using the Data Mapping Facility to convert your MFS Source to the required format maps and generate the HTML page in the image of your output MFS screen using the following steps:

Step 1: Generate Format Maps

Step 2: Generate the HTML

Step 3: Build an /#EXECIMS rule

Step 4: Generate the IMS Default HTML

See Chapter 13, “Data Mapping Facility,” for more information on using the Data Mapping Facility.

Step 1. Generate Format Maps

Before using the Data Mapping Facility to generate your format maps, you must know the following:

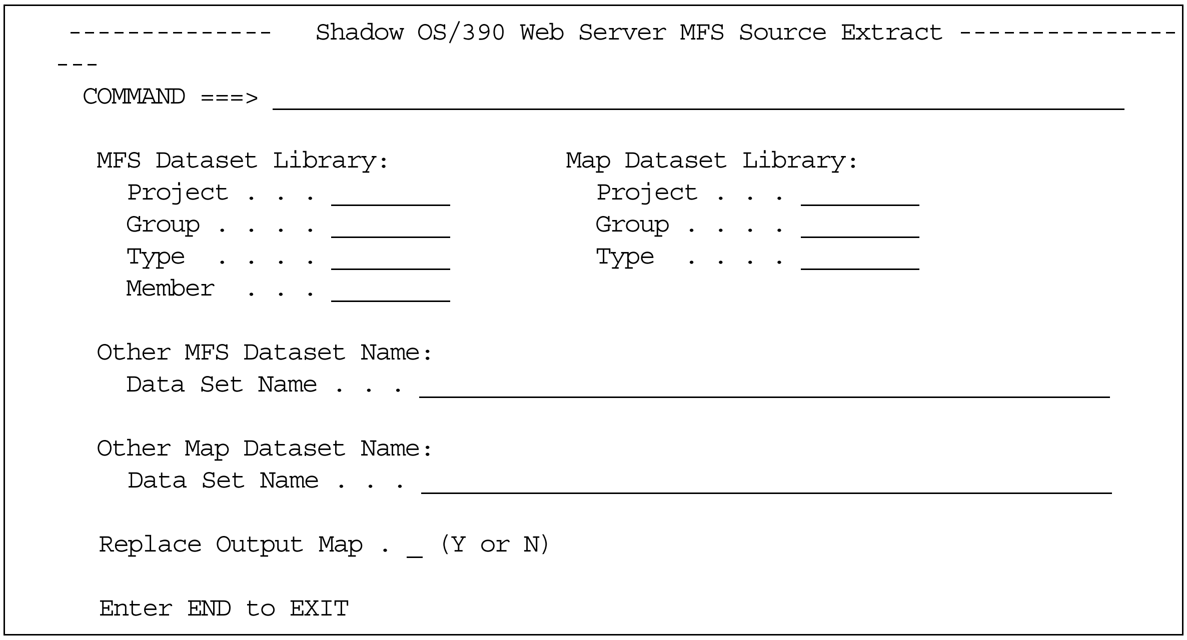
- **The name of your map dataset.** This is the dataset assigned to the **“**`SWSMAPP`**”** ddname in Shadow OS/390 Web Server’s startup JCL.
- **The name of your MFS Source library.** Contact your IMS System Administrator if you do not know the name of this library.

Using the Data Mapping Facility

In order to generate the input and output format maps from your MFS source:

1. Select the Data Mapping Facility from the Primary Option Menu.
2. Select the EXTRACT option from the Mapping Facility panel.
3. Select the Extract MFS option from the Mapping Facility panel.

The following screen appears:



```
 --------------   Shadow OS/390 Web Server MFS Source Extract ---------------
---
  COMMAND ===> ___________________________________________________________

   MFS Dataset Library:              Map Dataset Library:
     Project . . . ________            Project . . . ________
     Group . . . . ________            Group . . . . ________
     Type  . . . . ________            Type  . . . . ________
     Member  . . . ________

   Other MFS Dataset Name:
     Data Set Name . . . _______________________________________________

   Other Map Dataset Name:
     Data Set Name . . . _______________________________________________

   Replace Output Map . _ (Y or N)

   Enter END to EXIT
```

Figure 12–1. The Extract MFS panel from the Data Mapping Facility

4. Enter the following information on the Extract MFS panel:

   - Your MFS Source library name. This is the "`Source Library`".
   - Shadow OS/390 Web Server's Map dataset name. This is the "`Map Library`".

   The MFS Source member name can also be a member name mask. To extract all members in the source library, enter an asterisk (*). To see the member selection list, either:

   - Enter a member name mask (such as A*).
   - Do not enter a member name specification.

   The member selection list is created by browsing each member in the MFS source library. If a MSG, FMT or COPY statement is found in the member, the member is added as a selectable entry. However, if a member is the subject of a copy member in another member, it will be removed from the selection list.

   The replace option allows you to indicate that you do (Y) or do not (N) want pre-existing data map members to be replaced within the Data Map library.

5. Make the Maps Available to the server.

   Once you have extracted all the members you want to, press <PF3> (two times) in order to back up to the Shadow Server Mapping Facility Menu. At this screen, select the `Map Refresh` option. This makes your format maps available to Shadow OS/390 Web Server.

Essentially, the extract generates the equivalent of a:

- Message Input Descriptor (MID).
- Message Output Descriptor (MOD).
- Device Output Format (DOF).

The Input Map (or MID)

The `Input Map` (or `MID`) is used to format data from the HTML page for input to a specific transaction or command. Each HTML page must contain a query variable named “`SWSINMAP`” which points to an extracted map. The `Input Map` has a pointer to the `Output Map`. By default, this `Output Map` will be used to parse the transaction output. If you install the NEON supplied IMS LU 6.2 User Edit Exit (`DFSLUEE0`), the `Output Map` (`MOD`) specified by the IMS transaction will be used instead.

The Output Map (or MOD)

The `Output Map` (or `MOD`) is used to parse the output from the IMS transaction and build SQL Column Names. These column names are the same as those specified in the MFS Source. The `Output Map` contains a pointer to the Output Screen.

The Output Screen (or DOF)

The `Output Screen` (or `DOF`) is used to define the literals displayed in the HTML page as well as the placement of the SQL Column data supplied by the online IMS transaction. The Output Screen Map contains the name of the associated HTML dataset and member.

Step 2. Generate the HTML

The objective of this facility is to generate the HTML in the image of your output MFS screen. This is accomplished by formatting the HTML from the “`DEV`” (or Device Output Format (DOF)) portion of the MFS Source.

Before using the Data Mapping Facility, you must know the following:

- **The name of your map dataset.** This is the dataset assigned to the `“SWSMAPP”` ddname in Shadow OS/390 Web Server’s startup JCL.

- **The name of your output HTML Source library.** You may create your own dataset with the following specifications:

  – Record Format is VB
  – Record Length is 19036
  – Block Length is 19040
  – Dataset Organization is PO

Using the Data Mapping Facility

In order to generate HTML from your format maps:

1. Select the Data Mapping Facility from the Primary Option Menu.
2. Select the HTML Generation option.
3. Select the IMS HTML option.

   The following panel appears:

```
 -------------   Shadow OS/390 Web Server IMS HTML Generation -------------
----

 COMMAND ===> __________________________________________________________

   Map Dataset Library:            HTML Dataset Library:
     Project . . . ________          Project . . . ________
     Group . . . . ________          Group . . . . ________
     Type  . . . . ________          Type  . . . . ________
     Member  . . . ________          Member  . . . ________

   Other Map Dataset Name:
     Data Set Name . . . ______________________________________________

   Other HTML Dataset Name:
     Data Set Name . . . ______________________________________________

   Replace HTML member. _ (Y or N)
   HTML Heading . . . __________________________________________________

   Enter END to EXIT
```

Figure 12–2. The IMS HTML Generation panel

4. Enter Data on the Gen HTML panel. Enter:

   - Your HTML Source library name as the “`HTML Library`”.
   - Shadow OS/390 Web Server’s map dataset name.

   HTML can only be generated from MFS screen images or DOF’s (Device Output Formats). For this reason, you can only specify mapping dataset member names that represent screen types.

   To generate HTML for all “screen” type members within the data map dataset, enter an asterisk (*). To see the member selection list, either:

   - Enter a member name mask (such as A*).
   - Do not enter a member name specification.

Note:
The HTML member name is only useful when specifying a single data map member name. If you specify any type of mask that results in a selection list, the data map member name is used as the HTML member name.

The HTML replace option allows you to indicate that you do (Y) or do not (NO want pre-existing HTML sourcee members to be replaced within the HTML library.

The HTML heading allows you to specify a title for the HTML page that is created. This title is not reflected on the browser when using the default AutoHTML environment. The default title is specified in the HTML frameset found in `*.NEON.EXEC(IMS)`.

5. Make the Format Available to the server.

   Once you have generated all the members you want press <PF3> (two times) in order to back up to the Shadow OS/390 Web Server Mapping Facility Menu. At this screen, you need to select the `Map Refresh` option. This makes your format maps available to Shadow OS/390 Web Server.

Step 3. Build an /*EXECIMS rule

If you need special processing for your IMS transaction, build a `/*WWW` rule using an `/*EXECIMS` section.

How it Works

The `/*EXECIMS` section expects the input URL to contain query variables that match the names of field names within the Input Map. It also expects to have a query variable (`SWSINMAP`) passed on the input URL indicating which Input Map to use in order to build the input transaction.

NEON Systems ships a standard IMS entry point named:

```
.../NEON/IMS
```

This entry provides an interface that allows the user to enter an IMS transaction or IMS command.

Pressing <ENTER> on this page executes another Neon supplied rule which runs your IMS commands and transactions. The second rule, which is also supplied by neon, is:

```
.../NEON/IMSENTRY
```

With these two rules, the IMS LU 6.2 User Edit Exit (`DFSLUEE0`) installed and HTML generated by Shadow OS/390 Web Server’s Mapping Facility, you should be able to run the majority of your IMS transactions.

Formatting /*EXECIMS Section

The format of the /*EXECIMS section is:

```
/*EXECIMS     INPUTMAP(( IMS MESSAGE INPUT FORMAT )
              OUTPUTMAP( IMS MESSAGE OUTPUT FORMAT )
              PARTNERLUNAME( IMS APPC LUNAME )
              LOCALLUNAME( IMS LOCAL LUNAME )
              MODE( VTAM LOG MODE )
              SYMBOLICNAME( SYMBOLIC DESTINATION NAME )
              USERID(  USERID FOR SECURITY )
              PASSWORD( PASSWORD ASSOCIATED WITH USERID )
              PROFILE( SECURITY PROFILE )
              SECURITY( NONE | SAME | PROGRAM )
              CONNECTIONTYPE( IMS | IMC )
              OUTPUTHTML( DDNAME( DDNAMEVALUE )
                          DSNAME( DSNAMEVALUE )
                          MEMBER( MEMBERNAME ))
              ERRORURL( AUXILARY ERROR RULE )
```

Parameters

Parameter Name	Description
INPUTMAP	This is the input map name stored within the Shadow OS/390 Web Server Mapping Facility dataset. **Note:** This parameter does not override the INPUTMAP query variable passed on the input URL.
OUTPUTMAP	This is the output map name stored within the Shadow OS/390 Web Server Mapping Facility dataset. **Note:** This parameter does not override the specification from the IMS LU 6.2 User Edit Exit (DFSLUEE0).
PARTNERLUNAME	This is the APPC LU Name of the IMS System you want to execute the command or transaction. This can be ascertained from the “**/DIS A**.” IMS command or from your IMS System Administrator.
LOCALLUNAME	The name of the Local LU from which the caller's request is to originate.
MODE	This is the VTAM Log Mode to assign to the LU 6.2 connection. The log mode defines the VTAM session characteristics to the conversation.
SYMBOLICNAME	This is the symbolic name representing the Partner LU, the Partner TP_NAME and the MODE name for the session.
USERID	This is the userid to associate with all transactions or commands submitted using this rule.
PASSWORD	This is the password associated with the aforementioned userid.
PROFILE	This is the security profile to associate with the user assigned to transactions or commands submitted using this rule.

Table 12–1. Valid Parameters for the /*EXECIMS Section

Parameter Name	Description
SECURITY	This is the security level to associate with commands and or transactions submitted using this rule. The SECURITY parameter has three options: • **NONE**. Doesn't pass any security information to IMS/APPC during the execution of the transaction or command. This may be an acceptable option for non-production IMS environments or environments where security is controlled through application program interfaces rather than IMS system interfaces, like AGN, RACF, or CA-ACF2. • **SAME**. Uses the security information associated with the user that signed onto the Shadow OS/390 Web Server and passes it along with the IMS transaction. IMS is then able to authorize the transaction or command based upon IMS system level security established during the IMS generation process. • **PROGRAM**. Indicates that the userid and password will be passed from the executing program. This would be controlled through the use of the USERID and PASSWORD parameters.
CONNECTIONTYPE	This is the security level to associate with commands and or transactions submitted using this rule. The CONNECTIONTYPE parameter has two options: • IMS for non-conversational IMS transactions. • IMC for conversational IMS transactions. This option maintains the APPC conversation until the conversation is DEALLOCATED by IMS.
OUTPUTHTML	This option allows you to specify an HTML page to use to display the response from the IMS transaction or command. • **DDNAME**. This is the DDNAME from which to obtain the HTML output. When specified with the DSNAME subparameter, it specifies the DDNAME to associate with the dataset specified by the DSNAME parameter. When specified without the DSNAME subparameter, it specifies a DDNAME allocated via the Shadow OS/390 Web Server startup JCL. • **DSNAME.** This is the dataset name containing the HTML output. • **MEMBER.** This is the member name within the dataset associated with the DSNAME or DDNAME sub-parameter. When using the OUTPUTHTML parameter, you must specify a complete path to the dataset and member containing the HTML page. Nothing is defaulted nor assumed.
ERRORURL	This is used to specify an alternate Auxiliary Error URL used to process the error; it replaces the Neon supplied URL of `SYSTEM/ERROR/AUX`.
STARTUPURL	This is used to specify a URL where the user request is to be redirected when the user presses the "CLEAR" button. This overrides the NEON supplied rule of "`.../NEON/IMSENTRY`".
EXITURL	This is used to specify a URL where the user request is to be redirected when the user enters a "**/RCL**" IMS Command. This overrides the NEON supplied rule of "`.../NEON/IMSEXIT`".

Table 12–1. Valid Parameters for the /*EXECIMS Section

Example:

When a user logs onto IMS using a normal 3270 display station, IMS provides a clear screen which allows the user to enter commands or transactions. Neon supplies a rule "`.../NEON/IMS`" which provides the user the ability to enter IMS commands or transactions. The following is the text of the rule:

```
/*WWW /NEON/IMS AUTHREQ(NO)
*
*    IMS APPC ENTRY POINT
*       SEND IMS ENTRY SCREEN
*
/*FILE DATATYPE(INLINE) HTX(YES)

<HTML><HEAD>
<TITLE> NEON APPC/IMS INTERFACE </TITLE>
</HEAD><BODY BGCOLOR=000000>
<FONT COLOR=00F500>
<FORM ACTION="/NEON/IMSENTRY" METHOD=POST>
<PRE>
<BR> DATE: <%DATE(N)%>
<BR> TIME: <%TIME(N)%>
<BR> USER: <%WWW.USERID%>
<BR>
<BR> ENTER IMS COMMAND OR TRANSACTION CODE
<BR>
<BR> <INPUT NAME=IN1 TYPE=TEXT SIZE=119>
<BR>
</PRE>
<INPUT TYPE=HIDDEN NAME=SWSINMAP VALUE="DFSMI1">
<INPUT TYPE=HIDDEN NAME=SWSCNVID VALUE="0000000000000000">
<INPUT TYPE=HIDDEN NAME=PFKIN VALUE="ENTER">
</FORM>
</FONT>
</BODY></HTML>
```

This rule displays the HTML page with a single entry line in which the users enter their IMS Command or transaction code. By pressing the <ENTER> key, they start the main IMS transaction/command processing rule `.../NEON/IMSENTRY`. By using the defaults, this rule processes non-conversational, IMS transactions.

Step 4. Generate the IMS Default HTML

IMS ships several default MFS screens. Most of these are used to display system data upon various types of devices. Some of these are used as defaults for outbound messages where the transaction did not specify a MODNAME. NEON has supplied default source that **must** be run through the MFS Extract and HTML Generation process before any IMS commands or transactions may be executed. The following source needs to be extracted:

```
hlq.SAMP(IMSENTRY)
hlq.SAMP(IMSMOD)
```

Once extracted, the following `DOF` names must have HTML generated:

```
DFSDF2
DFSDOF1
```

CHAPTER 13: Data Mapping Facility

Data mapping allows Shadow OS/390 Web Server to map data from various sources. The term "mapping" indicates that Shadow OS/390 Web Server maintains the characteristics of data columns within rows of logical or physical data.

How It Works

Data maps are created with a series of ISPF panels that allow the user to specify a dataset containing a listing of a program that contains a data definition. A data definition in COBOL is a file or data definition; for PL/I, it is a DCL statement. The information (such as, length, format, type, and offset) about each field element is extracted from the data definition and then made available to Shadow OS/390 Web Server.

Clients of Shadow OS/390 Web Server can use the data maps to manipulate or view the logical or physical data.

Restrictions

Data mapping does not support "`OCCURS`" clauses which contain the "`DEPENDING ON`" clause. Whenever the "`OCCURS`" clause is used, it appends a numeric suffix to the corresponding column. For example, if you extracted the following on: "`FIELD-A`"

```
05    FIELD-A occurs 3 times
```

you would see these column names:

```
FIELD-A-1
FIELD-A-2
FIELD-A-3
```

Getting Started

A limited subset of the Shadow Event Facility (SEF) is provided to support data mapping.

1. Familiar yourself with the function provided in the panels.
2. Verify Shadow OS/390 Web Server is active.
3. Create the map.
4. View the map for correctness.
5. Modified if necessary, using the ISPF panels.

Maps are managed from a map library assigned to the Shadow OS/390 Web Server started task.

Recommendations

We recommend the following:

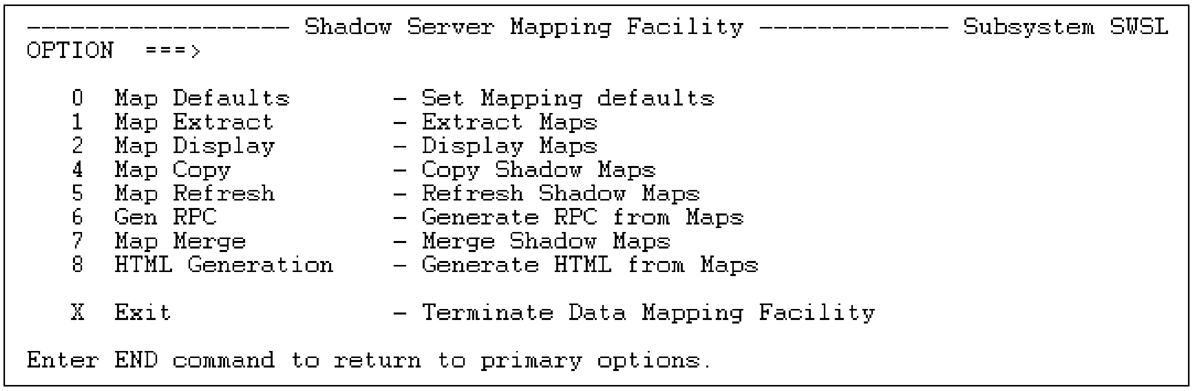
- Use one server as a “test” server and a second server as a "production" server.
- Use the DD statement `'SWSMAPP'` as part of initial setup to identify the dataset that contains the maps for your production server.
- For each server, allocate one or more datasets, as needed. To facilitate central control of the production map dataset, allocate a "staging" dataset for interim maps.

Data Mapping Checklist

- Identify the dataset that contains the compiler listings.
- Allocate a mapping dataset.
- Bring up the data mapping selection menu.
- Perform a Map Default to create the default settings for the library that will contain the user defined data maps.
- Perform a Map Extract to create the data map from compiler listings.
- Perform a Map Refresh to load the newly created map into the server.
- Perform a Map Display to verify that the map extraction completed correctly.
- Select Map Copy, Gen RPC, or Map Merge, as needed.
- Give the END command to return to the Shadow OS/390 Web Server main panel.

The ISPF Panels

Select Data Mapping from the Shadow OS/390 Web Server Primary Options Menu. The following panel appears:



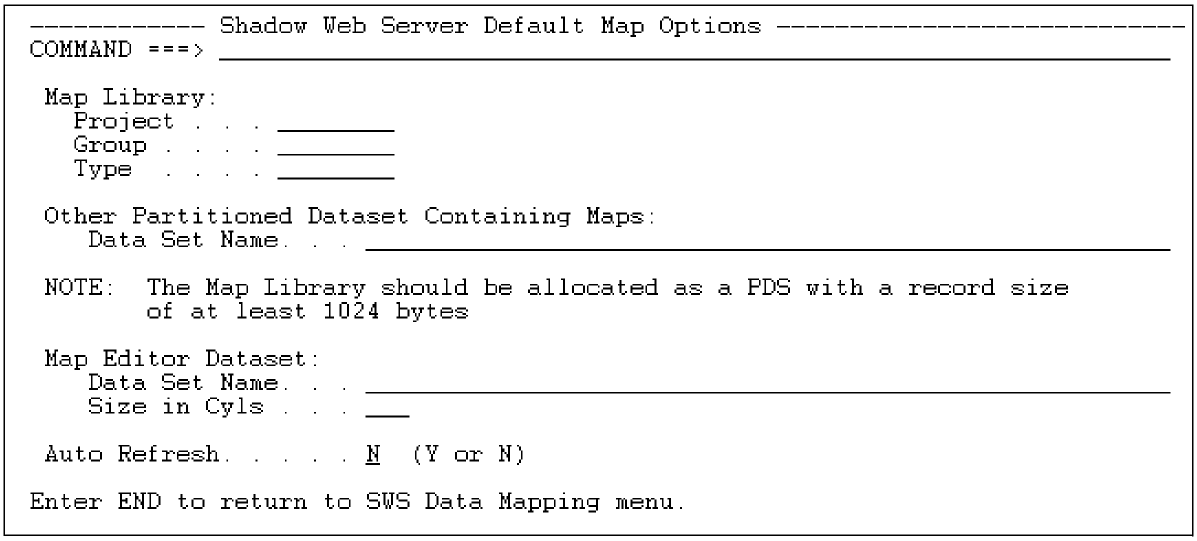
```
------------------ Shadow Server Mapping Facility ------------- Subsystem SWSL
OPTION  ===>

   0  Map Defaults       - Set Mapping defaults
   1  Map Extract        - Extract Maps
   2  Map Display        - Display Maps
   4  Map Copy           - Copy Shadow Maps
   5  Map Refresh        - Refresh Shadow Maps
   6  Gen RPC            - Generate RPC from Maps
   7  Map Merge          - Merge Shadow Maps
   8  HTML Generation    - Generate HTML from Maps

   X  Exit               - Terminate Data Mapping Facility

Enter END command to return to primary options.
```

Figure 13–1. The Data Mapping Facility Panel

Map Defaults

The Map Defaults panel allows for the default setting of the library that contains user defined data maps. The library must:

- Be previously allocated as a partitioned organized (PO) dataset.
- Have a logical record length (LRECL) of at least 1024 bytes. Other information, such as size and number of directory blocks, is usage dependent.



```
------------ Shadow Web Server Default Map Options ----------------------------
COMMAND ===> ________________________________________________________________

 Map Library:
   Project . . . ________
   Group . . . . ________
   Type  . . . . ________

 Other Partitioned Dataset Containing Maps:
    Data Set Name. . . _______________________________________________________

 NOTE:  The Map Library should be allocated as a PDS with a record size
        of at least 1024 bytes

 Map Editor Dataset:
    Data Set Name. . . _______________________________________________________
    Size in Cyls . . . ___

 Auto Refresh. . . . . N  (Y or N)

Enter END to return to SWS Data Mapping menu.
```

Figure 13–2. The Default Map Options Panel

Entering Information

1. Enter the information in the Default Map panel.

   If Auto Refresh is set to yes, a Data Map and HTML Data Set refresh is automatically performed whenever you exit. This eliminates the need to manually select the Map Refresh option.

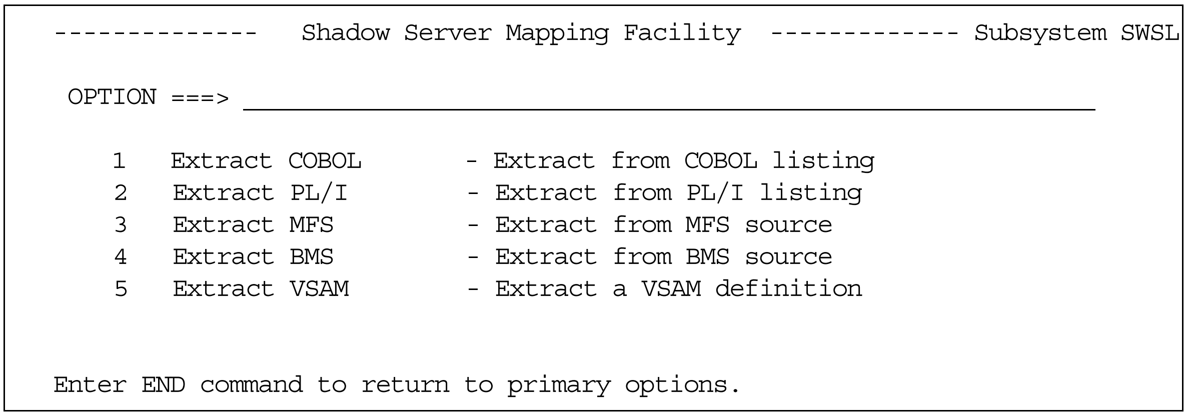
   Note:
   Auto Refresh can incur significant overhead if you have several changes to make and you exit after each change. We recommend that you either 1) turn-off Auto Refresh and use the Map Refresh option when finished, or 2) make all your changes before exiting.

2. Press <ENTER> after entering all the information on the panel.

   The message "`Profile Saved`" appears on the Status Line. This means the dataset name is saved in the ISPF user profile pool for Shadow OS/390 Web Server.

Map Extract

Select Map Extract from the Data Mapping Facility panel and the following panel appears:



```
 -------------   Shadow Server Mapping Facility  ------------- Subsystem SWSL

  OPTION ===> ________________________________________________________

     1   Extract COBOL       - Extract from COBOL listing
     2   Extract PL/I        - Extract from PL/I listing
     3   Extract MFS         - Extract from MFS source
     4   Extract BMS         - Extract from BMS source
     5   Extract VSAM        - Extract a VSAM definition


 Enter END command to return to primary options.
```

Figure 13–3. Shadow OS/390 Web Server Mapping Facility Panel

COBOL Listing Requirements
The COBOL program must have been compiled using the compiler options XREF(FULL) and MAP.

PL/I Listing Requirements

The PL/I program must be compiled using the compiler options XREF(FULL), MAP, AGGREGATE, and ATTRIBUTES(FULL).

Extract MFS

Extracts are done from the MFS source; it is not compiled.

Extract BMS

Extracts are done from the BMS source; it is not compiled.

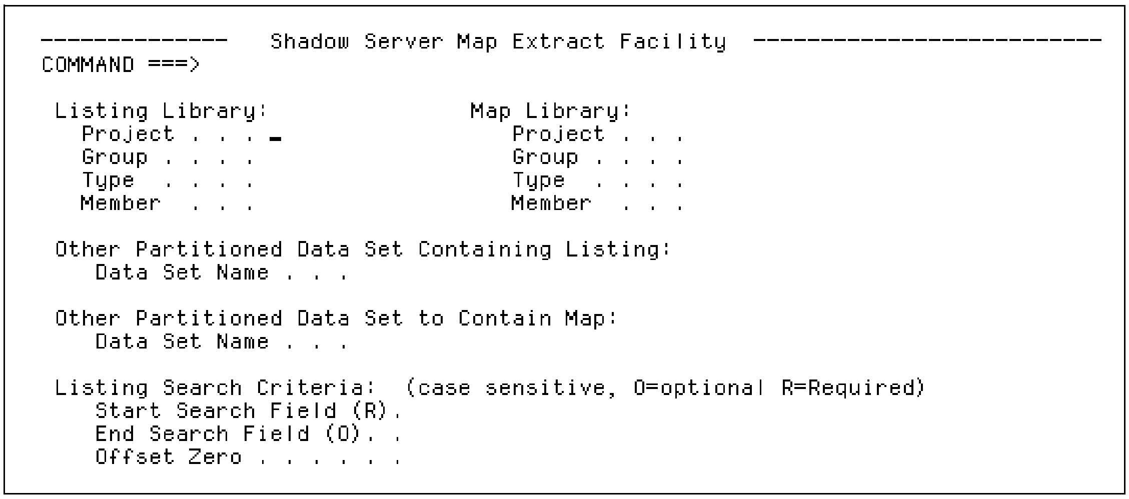
VSAM Listing Requirements

The VSAM program must be extracted using the COBOL or PL/I listing requirements.

Entering Information

- Select the program and press <ENTER>.

  The Map Extract Facility Panel appears.



```
--------------   Shadow Server Map Extract Facility  --------------------------
COMMAND ===>

 Listing Library:               Map Library:
   Project . . . _                 Project . . .
   Group . . . .                   Group . . . .
   Type  . . . .                   Type  . . . .
   Member  . . .                   Member  . . .

 Other Partitioned Data Set Containing Listing:
    Data Set Name . . .

 Other Partitioned Data Set to Contain Map:
    Data Set Name . . .

 Listing Search Criteria:  (case sensitive, O=optional R=Required)
    Start Search Field (R).
    End Search Field (O). .
    Offset Zero . . . . . .
```

Figure 13–4. The Map Extract Facility Panel

The Map Extract requires a listing dataset as input. The output from the extract is a data mapping definition that will be placed in the named map library, or you can specify another partitioned dataset for the listing and data map libraries.

The map library member name will be the name associated for this map by Shadow OS/390 Web Server.

Listing Search Criteria Start Search Field

This is used to search the listing dataset for the starting point of the language dependent data declaration. The search criteria must be unique enough to find the specific declaration to be mapped. For best results, use the full qualified name of the declaration as it appears in the listing.

End Search Field (optional)

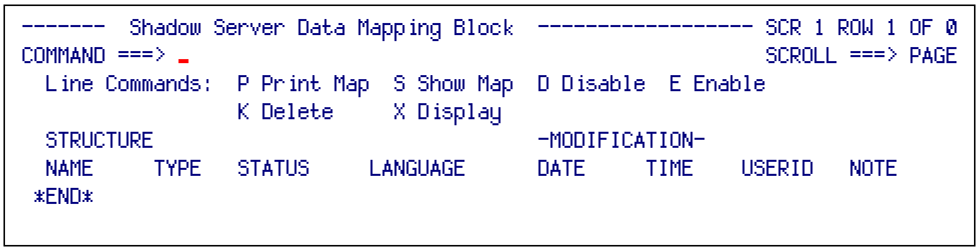
If this is left blank, extraction starts with the level number of the line found and continues until an equal or higher level is processed. If the field is not blank, extraction continues until the ending search string is found in the listing.

If the extract completes with no errors, a message "`Extract Successful`" appears in the upper right hand corner of the panel. At this point, both the map library and Shadow OS/390 Web Server contain the mapped structure definition.

The Offset Zero parameter indicates whether to set the Start Search Field offset to zero, even if it is not a group level or the first definition in a group.

Map Display

Select Map Display from the Data Mapping Facility panel and the following panel appears:



```
-------  Shadow Server Data Mapping Block  ------------------ SCR 1 ROW 1 OF 0
COMMAND ===> _                                                SCROLL ===> PAGE
  Line Commands:  P Print Map  S Show Map  D Disable  E Enable
                  K Delete     X Display
  STRUCTURE                                -MODIFICATION-
  NAME      TYPE   STATUS     LANGUAGE     DATE     TIME     USERID    NOTE
 *END*
```

Figure 13–5. The Data Mapping Block, Panel 1

This option displays existing data maps.

Structure Name

This corresponds to the member names within the map dataset.

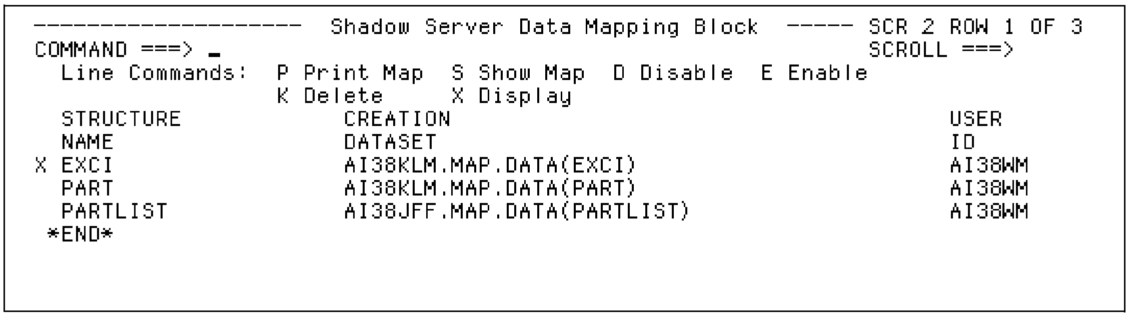
Type This corresponds to one of the following types of structure:

- ADABAS
- INPUT
- OUTPUT
- SCREEN
- LPTBL
- HEADER
- USER

The language, which is determined at the time of the extract, the creation date and time are only used for informational purposes. The extracted map is independent of language type.

Viewing the Next Panel

- Scroll to the right to view panel two.



```
--------------------  Shadow Server Data Mapping Block  ----- SCR 2 ROW 1 OF 3
COMMAND ===> _                                                SCROLL ===>
  Line Commands:  P Print Map  S Show Map  D Disable  E Enable
                  K Delete     X Display
  STRUCTURE           CREATION                                  USER
  NAME                DATASET                                   ID
X EXCI                AI38KLM.MAP.DATA(EXCI)                    AI38WM
  PART                AI38KLM.MAP.DATA(PART)                    AI38WM
  PARTLIST            AI38JFF.MAP.DATA(PARTLIST)                AI38WM
 *END*
```

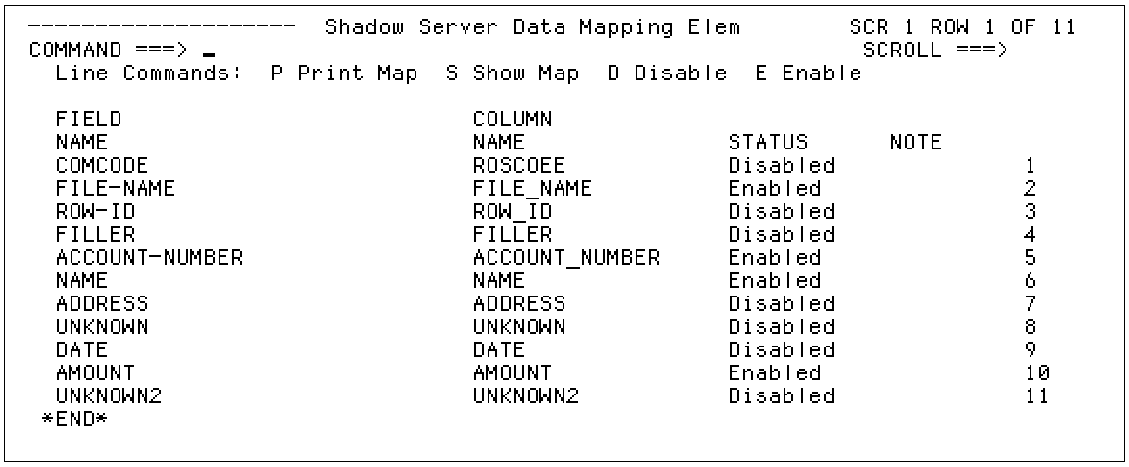
Figure 13–6. The Data Mapping Block, Panel 2

Creation dataset

This was used to create the extracted data map. The extractor’s user identification are displayed for informational purposes.

Viewing the Individual Data Elements

Use option *X* on any row of the map to view the individual data elements. In Figure 13–6, the structure EXCI was selected on the Map Display. The display shows each field name, its corresponding column name, and the status of each field.



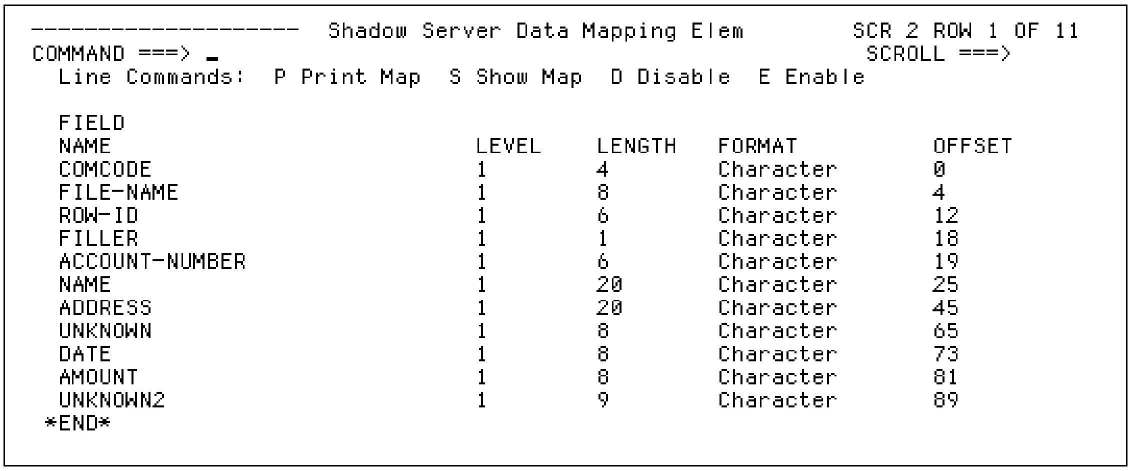
```
--------------------  Shadow Server Data Mapping Elem       SCR 1 ROW 1 OF 11
COMMAND ===> _                                              SCROLL ===>
  Line Commands:  P Print Map  S Show Map  D Disable  E Enable

  FIELD                          COLUMN
  NAME                           NAME              STATUS      NOTE
  COMCODE                        ROSCOEE           Disabled             1
  FILE-NAME                      FILE_NAME         Enabled              2
  ROW-ID                         ROW_ID            Disabled             3
  FILLER                         FILLER            Disabled             4
  ACCOUNT-NUMBER                 ACCOUNT_NUMBER    Enabled              5
  NAME                           NAME              Enabled              6
  ADDRESS                        ADDRESS           Disabled             7
  UNKNOWN                        UNKNOWN           Disabled             8
  DATE                           DATE              Disabled             9
  AMOUNT                         AMOUNT            Enabled              10
  UNKNOWN2                       UNKNOWN2          Disabled             11
 *END*
```

Figure 13–7. The Data Mapping Element, Panel 1

Fields	These can be enabled, disabled, and deleted in a similar fashion as the structure is.
Column	During Map Extract, column names were created using the field names and translating any dash characters to underscores. The Map Editor can be used to make column names more meaningful for users.

Viewing the Next Panel

- Scroll to the right to see the next panel:



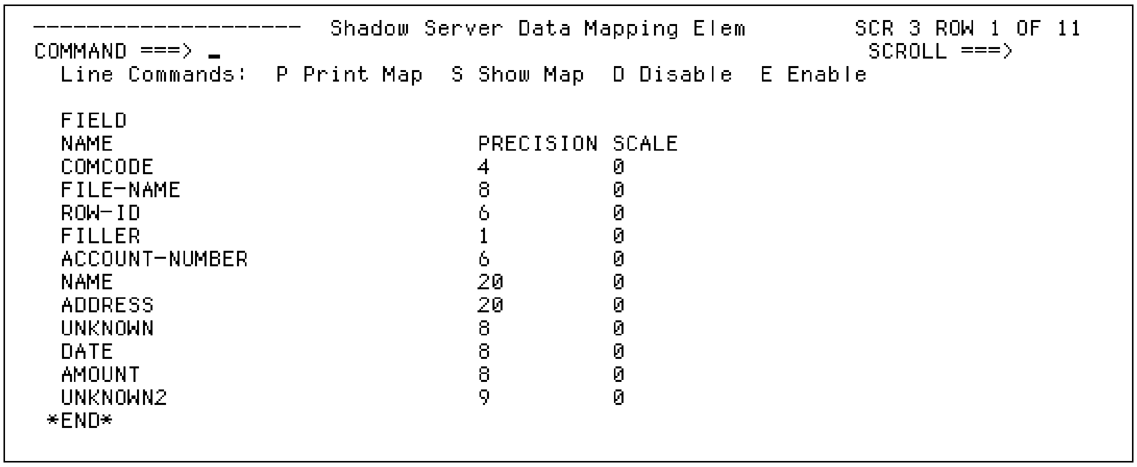
```
--------------------  Shadow Server Data Mapping Elem       SCR 2 ROW 1 OF 11
COMMAND ===> _                                               SCROLL ===>
  Line Commands:  P Print Map  S Show Map  D Disable  E Enable

  FIELD
  NAME                           LEVEL    LENGTH   FORMAT          OFFSET
  COMCODE                        1        4        Character       0
  FILE-NAME                      1        8        Character       4
  ROW-ID                         1        6        Character       12
  FILLER                         1        1        Character       18
  ACCOUNT-NUMBER                 1        6        Character       19
  NAME                           1        20       Character       25
  ADDRESS                        1        20       Character       45
  UNKNOWN                        1        8        Character       65
  DATE                           1        8        Character       73
  AMOUNT                         1        8        Character       81
  UNKNOWN2                       1        9        Character       89
 *END*
```

Figure 13–8. The Data Mapping Element, Panel 2

Field	This acts as a point of reference from panel to panel.
Level	This is maintained for informational purposes only.
Length	This is of primary importance in the map element.
Format	This is of primary importance in the map element. Various format types are character, binary, date, time, packed, decimal, and group.
Offset	This is of primary importance in the map element. An offset is maintained as the relative position 0 displacement from the beginning of the structure.

Viewing the Next Panel

- Scroll to the right to see the next panel. The field name is displayed along with the scale and precision of any decimal or packed data items.

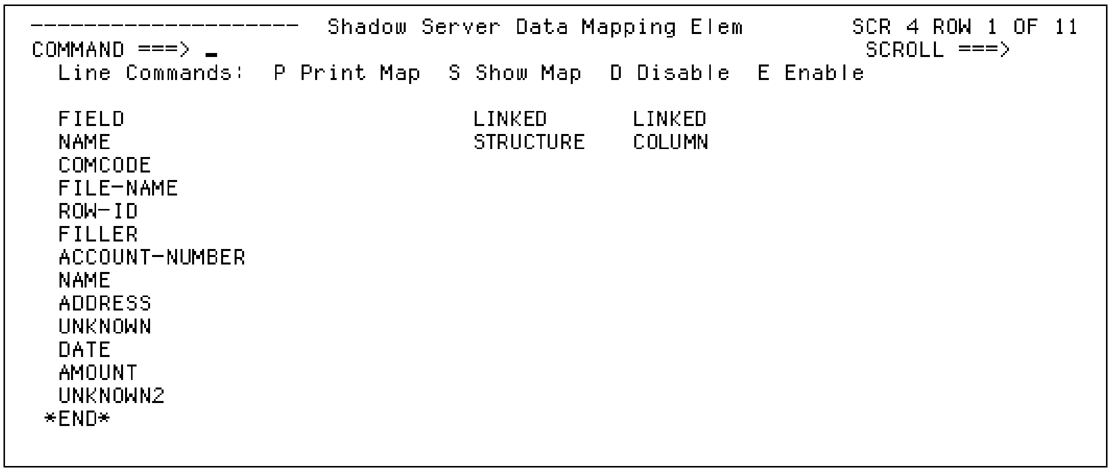


```
--------------------  Shadow Server Data Mapping Elem        SCR 3 ROW 1 OF 11
COMMAND ===> _                                                 SCROLL ===>
  Line Commands:  P Print Map  S Show Map  D Disable  E Enable

  FIELD
  NAME                          PRECISION SCALE
  COMCODE                       4         0
  FILE-NAME                     8         0
  ROW-ID                        6         0
  FILLER                        1         0
  ACCOUNT-NUMBER                6         0
  NAME                          20        0
  ADDRESS                       20        0
  UNKNOWN                       8         0
  DATE                          8         0
  AMOUNT                        8         0
  UNKNOWN2                      9         0
 *END*
```

Figure 13–9. The Data Mapping Element, Panel 3

Viewing the Next Panel

- Scroll to the right to see the next panel. The information on this panel is intended for future use.

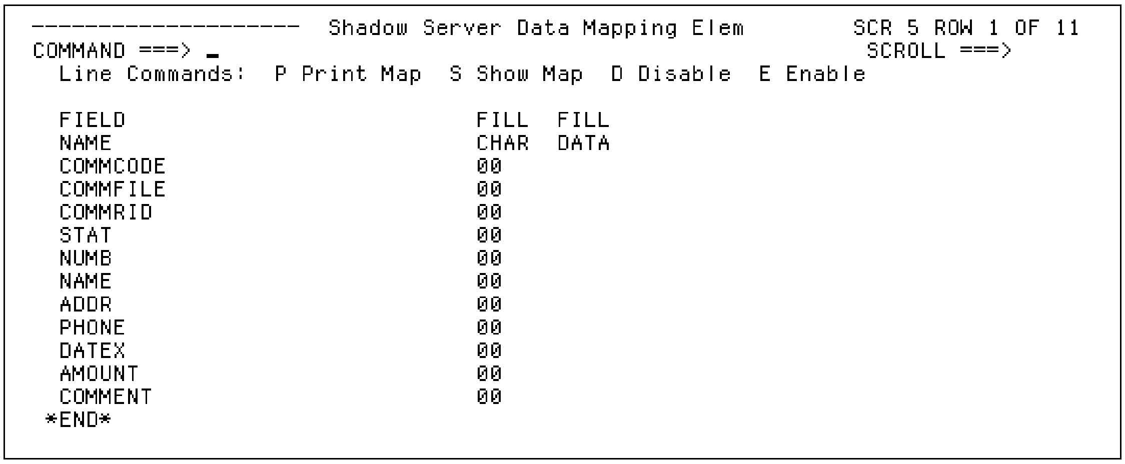


```
--------------------  Shadow Server Data Mapping Elem        SCR 4 ROW 1 OF 11
COMMAND ===> _                                                 SCROLL ===>
  Line Commands:  P Print Map  S Show Map  D Disable  E Enable

  FIELD                         LINKED       LINKED
  NAME                          STRUCTURE    COLUMN
  COMCODE
  FILE-NAME
  ROW-ID
  FILLER
  ACCOUNT-NUMBER
  NAME
  ADDRESS
  UNKNOWN
  DATE
  AMOUNT
  UNKNOWN2
 *END*
```

Figure 13–10. The Data Mapping Element Continued, Panel 4

Viewing the Next Panel

- Scroll to the right to see the next panel. The information on this panel is intended for future use.

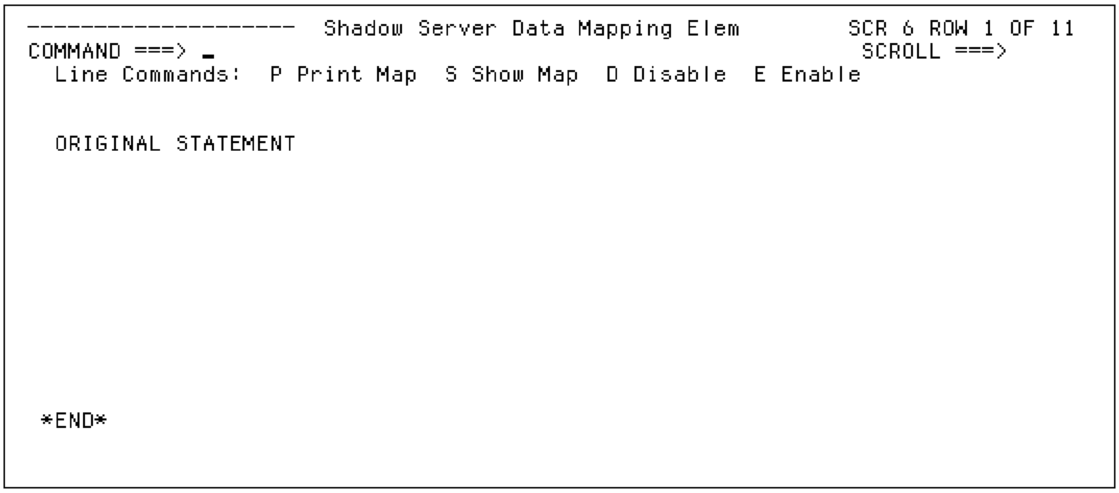


```
--------------------  Shadow Server Data Mapping Elem        SCR 5 ROW 1 OF 11
COMMAND ===> _                                                SCROLL ===>
  Line Commands:  P Print Map  S Show Map  D Disable  E Enable

  FIELD                          FILL  FILL
  NAME                           CHAR  DATA
  COMMCODE                       00
  COMMFILE                       00
  COMMRID                        00
  STAT                           00
  NUMB                           00
  NAME                           00
  ADDR                           00
  PHONE                          00
  DATEX                          00
  AMOUNT                         00
  COMMENT                        00
 *END*
```

Figure 13–11. The Data Mapping Element, Panel 5

Viewing the Next Panel

- Scroll to the right to see the next panel. It contains the original listing lines from which the elements were extracted. For items that where entered with the editor, these will not be available.

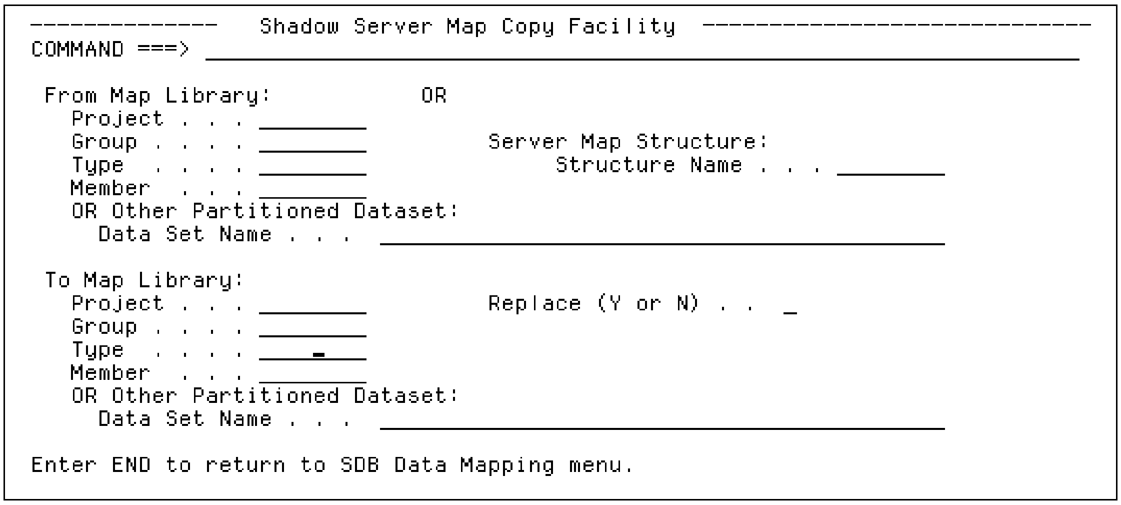


```
--------------------  Shadow Server Data Mapping Elem        SCR 6 ROW 1 OF 11
COMMAND ===> _                                                SCROLL ===>
  Line Commands:  P Print Map  S Show Map  D Disable  E Enable


  ORIGINAL STATEMENT











 *END*
```

Figure 13–12. The Data Mapping Element, Panel 6

Copy Map

The Map Copy function allows data maps to be copied from one map library to another or from SDF to a map library.



```
-------------   Shadow Server Map Copy Facility  ----------------------------
COMMAND ===> _________________________________________________________________

 From Map Library:           OR
   Project . . . ________
   Group . . . . ________         Server Map Structure:
   Type  . . . . ________              Structure Name . . . ________
   Member  . . . ________
   OR Other Partitioned Dataset:
     Data Set Name . . .  ____________________________________________

 To Map Library:
   Project . . . ________         Replace (Y or N) . .  _
   Group . . . . ________
   Type  . . . . ________
   Member  . . . ________
   OR Other Partitioned Dataset:
     Data Set Name . . .  ____________________________________________

Enter END to return to SDB Data Mapping menu.
```

Figure 13–13. The Map Copy Facility Panel

Refresh Map

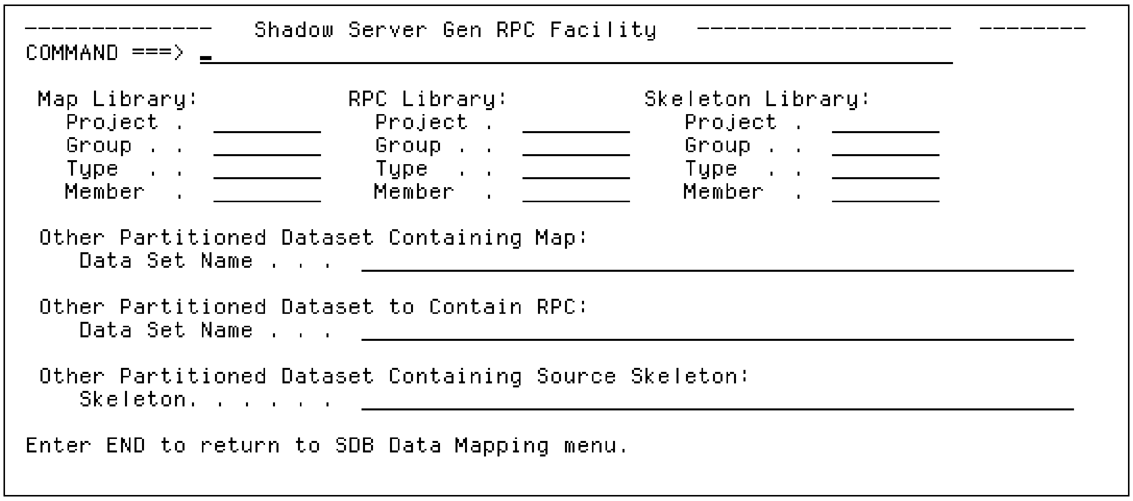
When used, Shadow OS/390 Web Server checks the library for modifications, and then refreshes the SDF in core map tables from the library. The message "`Refresh Successful`" appears on the mapping facility menu options if it completed without any errors.

The Shadow OS/390 Web Server Mapping Library

This library is assigned to DDNAME `'SWSMAPP'` in the started task JCL. If you are executing Shadow OS/390 Web Server in DEBUG mode, the DD can be allocated to TSO prior to starting the server.

Generate RPC

This option generates RPC programs from an extracted data map by generating the SQLBINDCOL statements into a new PDS member. It does this by using the skeleton program provided in the same PDS. The skeleton program contains all the language and application specific code required to perform the RPC task. Within the skeleton are keywords that are needed to substitute information and write the new specified member.



```
---------------   Shadow Server Gen RPC Facility   --------------------  --------
COMMAND ===> _

 Map Library:            RPC Library:          Skeleton Library:
   Project .  ________     Project .  ________    Project .  ________
   Group . .  ________     Group . .  ________    Group . .  ________
   Type  . .  ________     Type  . .  ________    Type  . .  ________
   Member  .  ________     Member  .  ________    Member  .  ________

 Other Partitioned Dataset Containing Map:
    Data Set Name . . .  ______________________________________________________

 Other Partitioned Dataset to Contain RPC:
    Data Set Name . . .  ______________________________________________________

 Other Partitioned Dataset Containing Source Skeleton:
    Skeleton. . . . . .  ______________________________________________________

Enter END to return to SDB Data Mapping menu.
```

Figure 13–14. The Generate RPC Facility Panel

Example

The following is an example of a skeleton COBOL program.

```
CBL  APOST
 010010 IDENTIFICATION DIVISION.
 010020 PROGRAM-ID.  DFSSAM02.
 010080 ENVIRONMENT DIVISION.
 010090 CONFIGURATION SECTION.
 010100 SOURCE-COMPUTER. IBM-370.
 010110 OBJECT-COMPUTER. IBM-370.
 010120 DATA DIVISION.
 010130 WORKING-STORAGE SECTION.
      COPY SBCPHD.
      77 SDF-RETURN-CODE   PIC S9(05) VALUE 0.
      77 STATEMENT-HANDLE        USAGE IS POINTER .
      77 SQL-PRECISION     PIC S9(5) COMP VALUE 0.
      77 SQL-SCALE         PIC S9(5) COMP VALUE 0.
      77 SQL-COLUMN-LEN    PIC S9(5) COMP VALUE 1.
      77 SQL-COLUMN-NAME-LEN PIC S9(5) COMP.
      77 SQL-COLUMN-NUMBER PIC S9(5) COMP.
      77 SQL-COLUMN-NAME   PIC X(30).
      77 ERROR-MESSAGE-AREA      PIC X(256) VALUE IS SPACES.
      77 TRACE-MESSAGE-AREA      PIC X(256) VALUE IS SPACES.
      77 STRING-PTR              PIC S9(5) COMP VALUE IS 1.
      77 CONNECTION-HANDLE       USAGE IS POINTER.
      77 ENVIRONMENT-HANDLE      USAGE IS POINTER.
      77 ERROR-MSG-LENGTH-AREA   PIC S9(5) COMP VALUE 0.
      77 NATIVE-ERROR-CODE-AREA  PIC S9(5) COMP VALUE 0.
      77 SQLSTATE-DATA-AREA      PIC X(6) VALUE IS SPACES.
      @DATABUFFER
 060110 LINKAGE SECTION.
 080010 PROCEDURE DIVISION.
 080020 INIT.
```

```
      @SQLBINDCOL BEGIN
      MOVE @LENGTH TO SQL-COLUMN-LEN.
      MOVE @COLUMN_NAME_LENGTH TO SQL-COLUMN-NAME-LEN.
      MOVE @COLUMN_NAME TO SQL-COLUMN-NAME.
      MOVE @SEQ TO SQL-COLUMN-NUMBER.
      MOVE @PRECISION TO SQL-PRECISION.
      MOVE @SCALE TO SQL-SCALE.
      CALL 'SDCPBC' USING STATEMENT-HANDLE
               SQL-COLUMN-NUMBER
               SQL-C-DEFAULT
               SQL-SMALLINT
               SQL-PRECISION
               SQL-SCALE
               SQL-NO-NULLS
               @FIELD_NAME
               SQL-COLUMN-LEN
               SQL-COLUMN-NAME
               SQL-COLUMN-NAME-LEN.
      MOVE RETURN-CODE TO SDF-RETURN-CODE.
      IF SQL-INVALID-HANDLE OR SQL-ERROR OR SQL-NO-DATA-FOUND
        PERFORM 0000-ERROR-ROUTINE
      END-IF.
      @SQLBINDCOL END
      CALL 'SDCPTH' USING STATEMENT-HANDLE SQL-THROW-DONE.
      MOVE RETURN-CODE TO SDF-RETURN-CODE.
      IF SQL-INVALID-HANDLE OR SQL-ERROR OR SQL-NO-DATA-FOUND
        PERFORM 0000-ERROR-ROUTINE THRU 0000-ERROR-EXIT
      END-IF.
 080140 EXIT-RTN.
 080160     GOBACK.
   0000-ERROR-ROUTINE.
      MOVE 256 TO SQL-PRECISION.
      IF SQL-INVALID-HANDLE GO TO 0000-ERROR-EXIT.
***************************************************************
*        IF AN ERROR OCCURS CALL THE SQLERROR ROUTINE
***************************************************************
      CALL 'SDCPSE' USING ENVIRONMENT-HANDLE CONNECTION-HANDLE
               STATEMENT-HANDLE SQLSTATE-DATA-AREA
               NATIVE-ERROR-CODE-AREA
               ERROR-MESSAGE-AREA
               SQL-COLUMN-LEN ERROR-MSG-LENGTH-AREA.
      MOVE RETURN-CODE TO WS-ODBCAPI-RETURN-CODE.
      IF SQL-SUCCESS OR SQL-SUCCESS-WITH-INFO
      PERFORM 0000-ERROR-DISPLAY-ROUTINE THRU
   0000-ERROR-DISPLAY-EXIT.
   0000-ERROR-EXIT.
   0000-ERROR-DISPLAY-ROUTINE.
***************************************************************
*    SEND THE ERROR MESSAGE TO THE CLEINT USING SQLRETURNSTATUS
***************************************************************
      STRING 'HOST ERROR MESSAGE - ' ERROR-MESSAGE-AREA
      DELIMITED BY SIZE INTO TRACE-MESSAGE-AREA WITH
      POINTER STRING-PTR
      END-STRING.
```

```
CALL 'SDCPRS' USING CONNECTION-HANDLE TRACE-MESSAGE-AREA
         SQL-NTS NATIVE-ERROR-CODE-AREA.
         0000-ERROR-DISPLAY-EXIT.
```

The Example Explained

- The statement:

  ```
  @DATABUFFER
  ```

  directs the facility to substitute the originally extracted information into the program at the location of this statement.

- The following statements declare the beginning and ending of the SQLBINDCOL substitution. All of the statements between the begin and end are replicated for the number of ENABLED fields in the map data.

  ```
  @SQLBINDCOL BEGIN
  @SQLBINDCOL END
  ```

- The following keywords can appear between the SQLBINDCOL BEGIN and SQLBINDCOL END statements. These keywords are substituted with the proper values for each ENABLED field in the data map.

  ```
  @LENGTH .    - the length of the field element
  @COLUMN_NAME_LENGTH  - the length of the column name.
  @COLUMN_NAME  - the column name used to identify the field
  @TYPE - SQL data type of column data. All DB2 SQL data types
  are supported except for graphic (DBCS) data.
  @SEQ - a sequentially assigned number for this column
  @PRECISION  - the precision of the field
  @SCALE  - the scale of the field
  @FIELD_NAME  - the field name itself as defined in the
  @DATABUFFER.
  ```

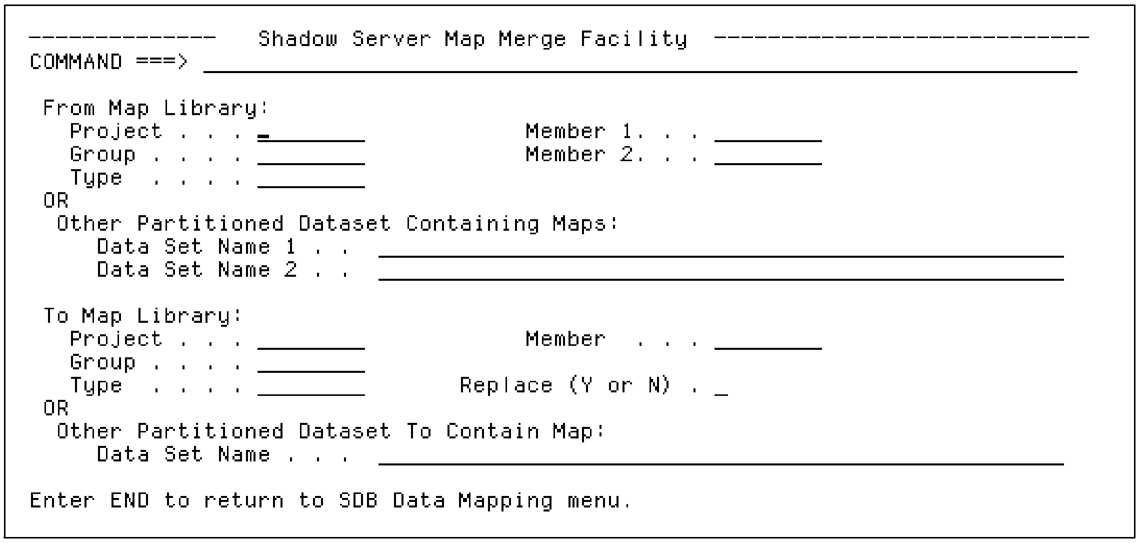
It should be noted that the skeleton can contain as many or as few statements as you want. You do not need to use all the keywords to have a complete program. For example, a skeleton member containing only:

```
@SQLBINDCOL BEGIN
@FIELD_NAME
@SQLBINDCOL END
```

would generate a list of ENABLED field names as defined in the data map.

Merge Maps

This option allows a data map to be concatenated (merged) to a second data map, resulting in a third (output) data map. The function recalculates the offsets of any merged items from `MEMBER 2` of the input map library and writes both `MEMBER 1` and `MEMBER 2` into the `MEMBER` specified in the "`To Map Library`".

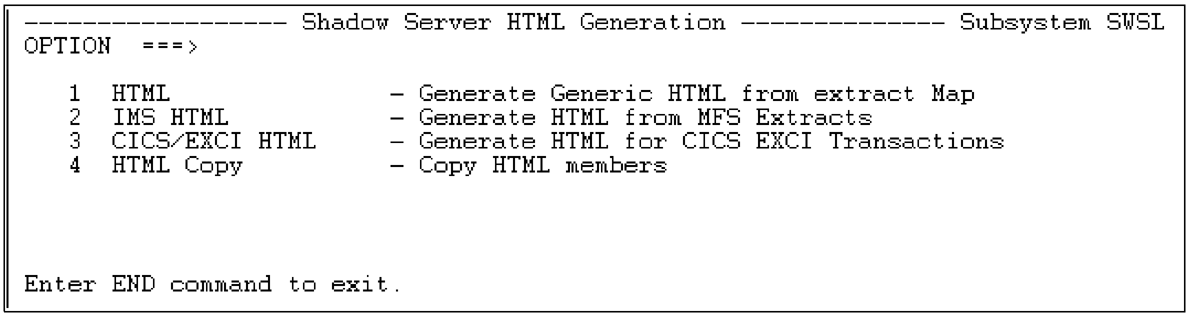


```
--------------   Shadow Server Map Merge Facility  ----------------------------
COMMAND ===> ________________________________________________________________

 From Map Library:
   Project . . . ________           Member 1. . . ________
   Group . . . . ________           Member 2. . . ________
   Type  . . . . ________
 OR
  Other Partitioned Dataset Containing Maps:
     Data Set Name 1 . .   ___________________________________________________
     Data Set Name 2 . .   ___________________________________________________

 To Map Library:
   Project . . . ________           Member  . . . ________
   Group . . . . ________
   Type  . . . . ________       Replace (Y or N) . _
 OR
  Other Partitioned Dataset To Contain Map:
     Data Set Name . . .   ___________________________________________________

Enter END to return to SDB Data Mapping menu.
```

Figure 13–15. The Map Merge Facility Panel

HTML Generation

The HTML Creation panel specifies 1) which map library contains the user defined maps and datasets and 2) which HTML library and datasets stores the results.

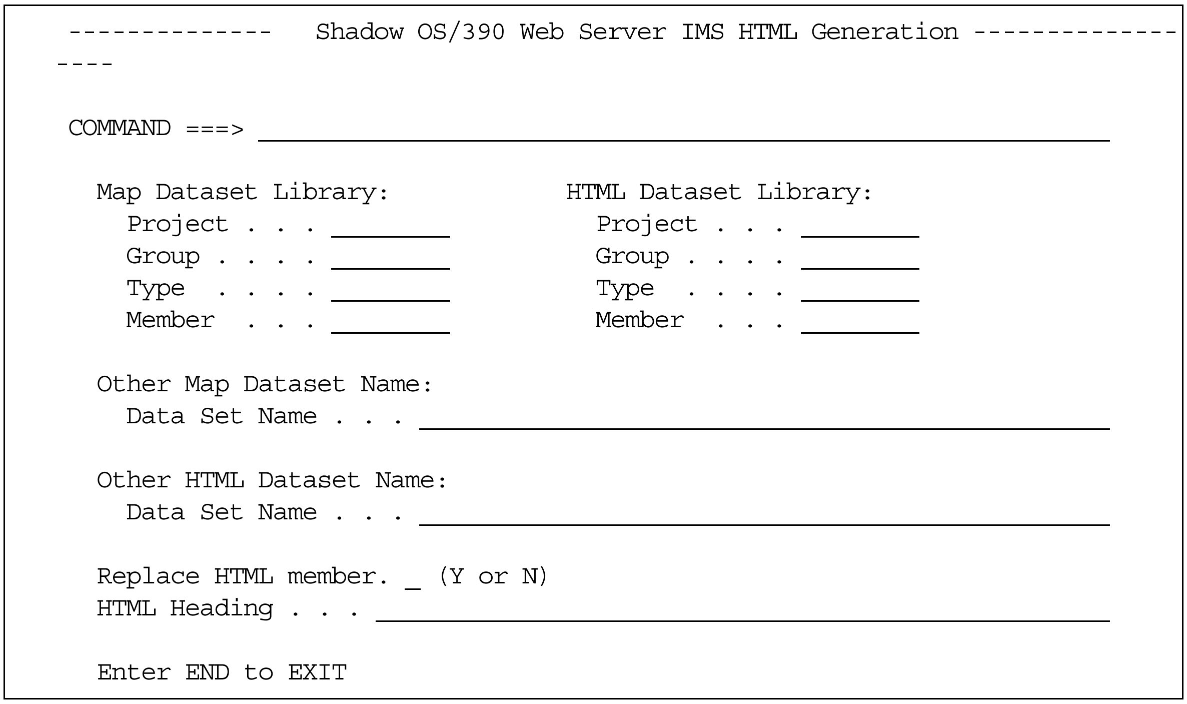


```
------------------ Shadow Server HTML Generation -------------- Subsystem SWSL
OPTION  ===>

   1  HTML                - Generate Generic HTML from extract Map
   2  IMS HTML            - Generate HTML from MFS Extracts
   3  CICS/EXCI HTML      - Generate HTML for CICS EXCI Transactions
   4  HTML Copy           - Copy HTML members




Enter END command to exit.
```

Figure 13–16. The HTML Generation Panel

Choose IMS HTML to create a Web browser image that has the same format (display of information) as the MFS source on the 3270 panel.

Note:
Before you can generate the HTML, the MFS data must be extracted and loaded into the map library, and the map refreshed before you can generate the HTML. Refer to Chapter 12, "AutoHTML - Web Enabling Transactions," for more information.



```
 -------------   Shadow OS/390 Web Server IMS HTML Generation -------------
----

 COMMAND ===> __________________________________________________________

   Map Dataset Library:              HTML Dataset Library:
     Project . . . ________            Project . . . ________
     Group . . . . ________            Group . . . . ________
     Type  . . . . ________            Type  . . . . ________
     Member  . . . ________            Member  . . . ________

   Other Map Dataset Name:
     Data Set Name . . . _______________________________________________

   Other HTML Dataset Name:
     Data Set Name . . . _______________________________________________

   Replace HTML member. _ (Y or N)
   HTML Heading . . . __________________________________________________

   Enter END to EXIT
```

Figure 13–17. The IMS HTML Generation Panel

Using Data Maps in Client Programs

The following are examples of the MAP parameter, used for CICS and IMS calls to SDF.

```
call shadow_cics('EXCI','EXCC','EXCI','DFH$AXCS',2,'FILEA   ',' 1','',120,'',
'MAP(NAME(EXCI) FIELDS(*))')
```

```
call shadow_ims('IMS','PART','IMSLU62','SAME','3007228','MAP(NAME(PART)
FIELDS(*) FORMAT(HORZ))')
```

```
call shadow_ims('IMS','PART','IMSLU62','SAME','*','MAP(NAME(PARTLIST) FIELDS(*)
FORMAT(VERT))')
```

MAP Subparameter	Description
NAME	This entry should correspond to the name assigned to the map during extraction.
FIELDS	There are two ways to return data from all columns that are enabled in the map definition: either use an asterisk after FIELDS (as shown), or leave out FIELDS altogether. To exclude some columns, enter the names of the enabled columns you do want returned in the parentheses after FIELDS.
FORMAT	This entry determines whether output will be oriented vertically or horizontally. Note that for a CICS call, FORMAT is not valid; for IMS, use `FORMAT(HORZ)` or `FORMAT(VERT)`.

Table 13–1. Map Subparameters for CICS and IMS Calls

For more information on parameters for CICS and IMS calls, see the Appendixes of the *Installation Guide*.

CHAPTER 14: Shadow ADABAS Server

The Shadow ADABAS Server provides a method of accessing existing ADABAS data from the desktop.

This new add-on component to NEON's Shadow products provide reliable, high-performance access to ADABAS data. The architectural flexibility offered by the Shadow ADABAS Server provides desktop client and Web browser access, as well as extends the life span and improves the investment return of existing ADABAS data.

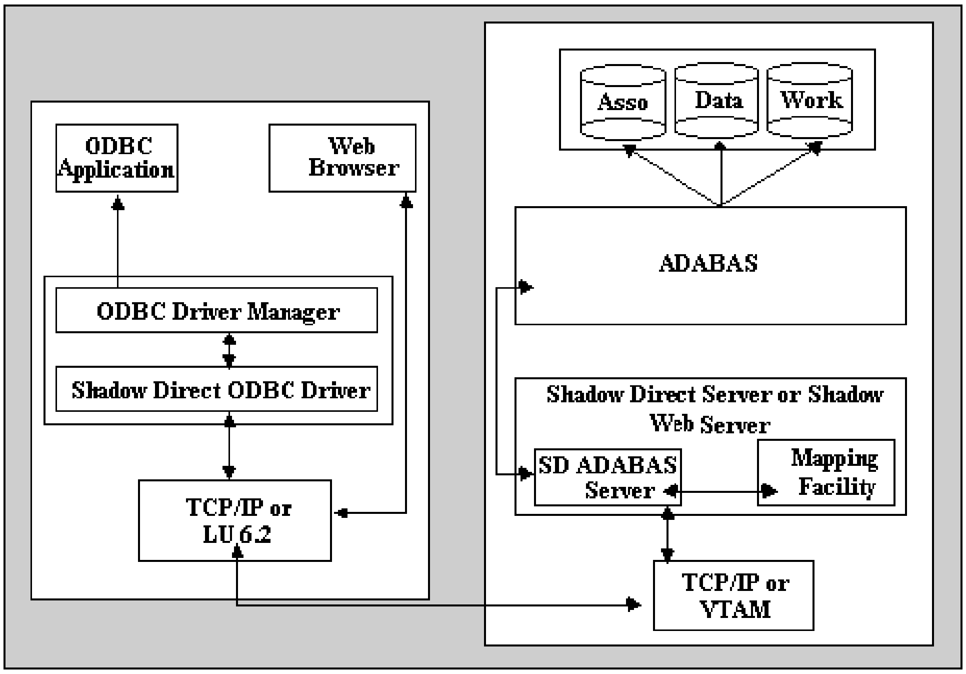
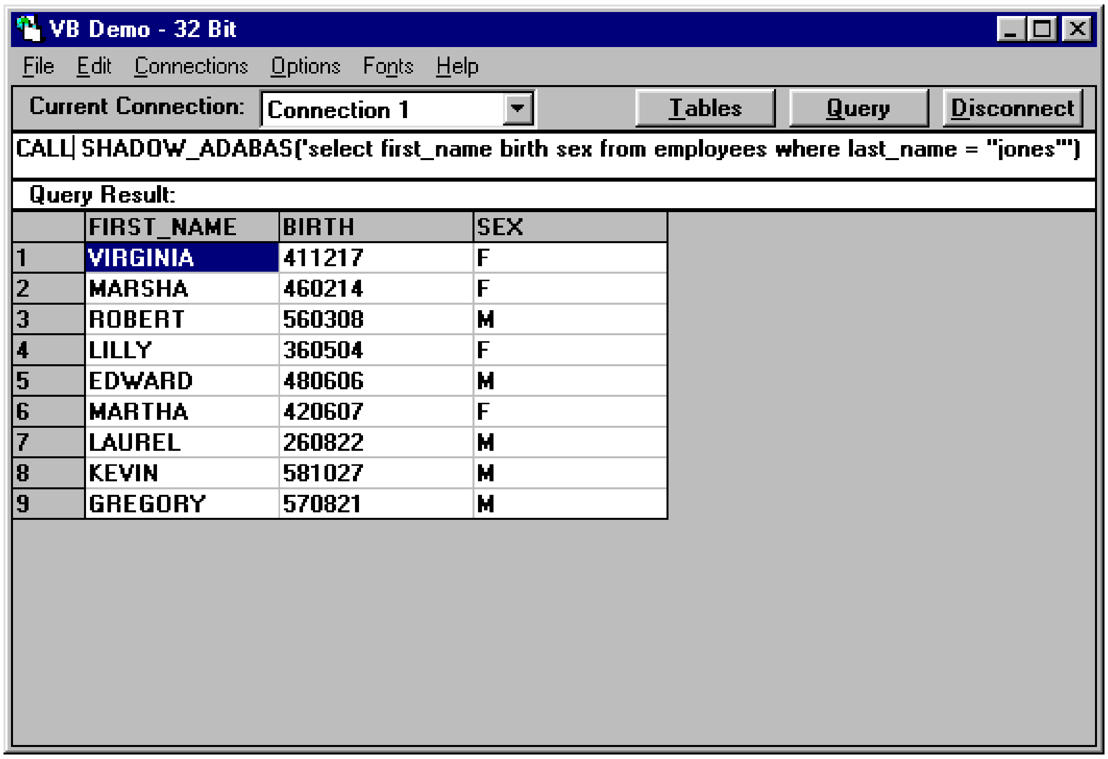


Figure 14–1. Shadow ADABAS Server Environment

How It Works

The Shadow ADABAS Server transforms a client SQL request into single or multiple ADABAS direct calls. The client requests arrive via the CALL SHADOW_ADABAS client interface.

The following example shows a SQL statement requesting the first name, birth date and age of all employees whose last name is Jones:



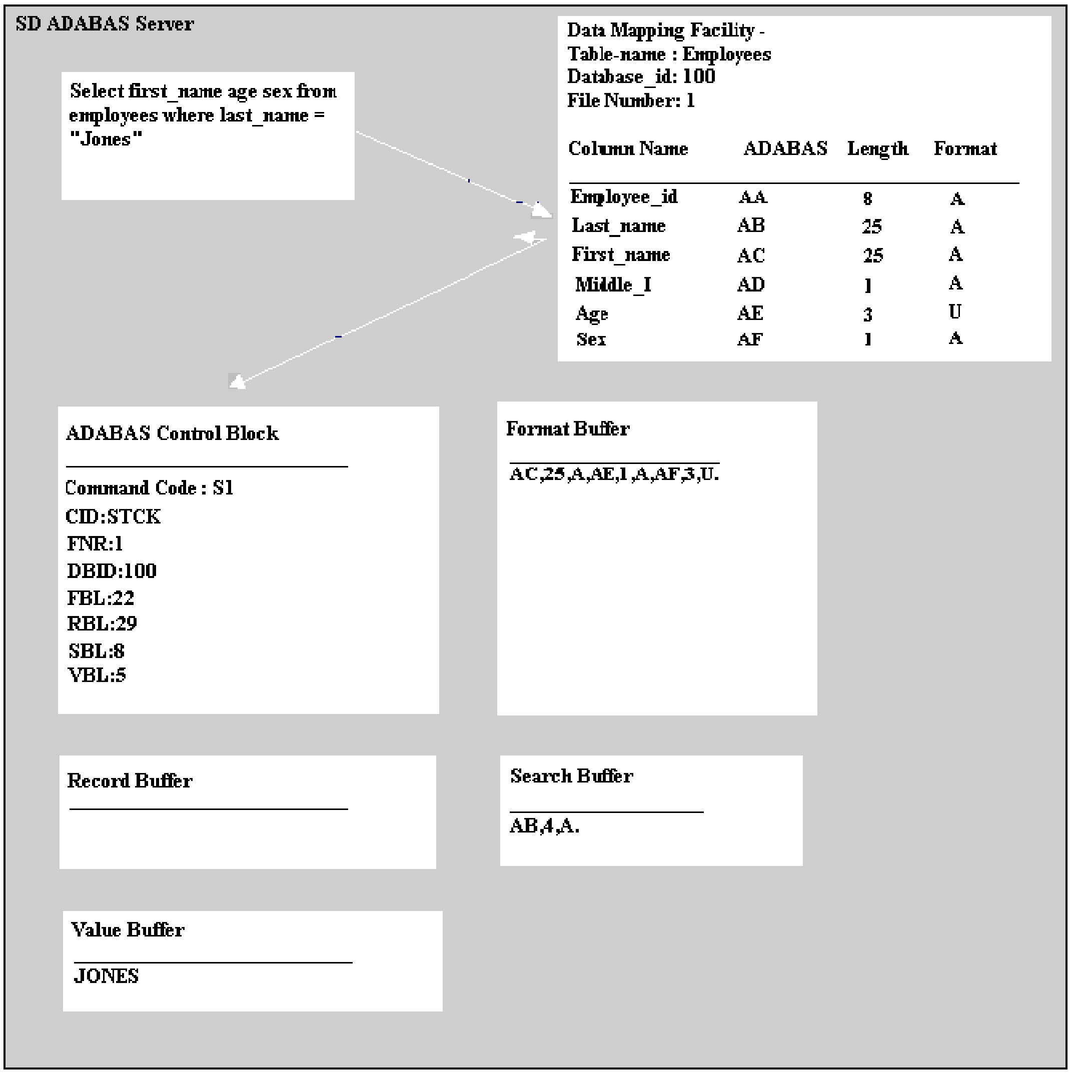
```
CALL SHADOW_ADABAS('select first_name birth sex from employees where last_name = "jones"')
```

	FIRST_NAME	BIRTH	SEX
1	VIRGINIA	411217	F
2	MARSHA	460214	F
3	ROBERT	560308	M
4	LILLY	360504	F
5	EDWARD	480606	M
6	MARTHA	420607	F
7	LAUREL	260822	M
8	KEVIN	581027	M
9	GREGORY	570821	M

Figure 14–2. SQL Statement Example

Note:
You must include "`CALL SHADOW _ADABAS`" as part of your SQL statement as shown in the example above.

Since ADABAS does not maintain a relational catalog describing information for every table, the Shadow Server maintains the information in the Shadow Data Mapping Facility. The proper ADABAS Control Block, Format Buffer, Record Buffer, Search Buffer, Value Buffer and ISN Buffer are created based on the SQL statement processing requirements as depicted in the following diagram:



Column Name	ADABAS	Length	Format
Employee_id	AA	8	A
Last_name	AB	25	A
First_name	AC	25	A
Middle_I	AD	1	A
Age	AE	3	U
Sex	AF	1	A

Figure 14–3. SD ADABAS Server

Shadow Mapping Facility

Packaged with the Shadow ADABAS Server are two utility programs, SDADEX and SDADDM. These programs extract the ADABAS file and field definitions, also known as maps, for import into the Shadow Data Mapping Facility. The definitions are used at execution time to formulate the information required to build the ADABAS direct calls.

SDADEX

SDADEX extracts information from the DDDRUCK output using an ADAREP report as input. The resulting data mapping statements represent all files and fields extracted from the ADAREP report.

The SDADEX extract program can (optionally) input ADAWAN definitions. The ADAWAN long field names are used for the long column names within the Shadow Mapping facility. If present, any MU and/or PE limiting specifications are also carried into the Shadow Mapping Facility. For example, an MU limit of 10 in PREDICT causes SDADEX utility to generate 10 unique column names for this ADABAS field, instead of the default of 191. Column names within Shadow are limited to 30 characters, even though PREDICT allows up to 32 characters.

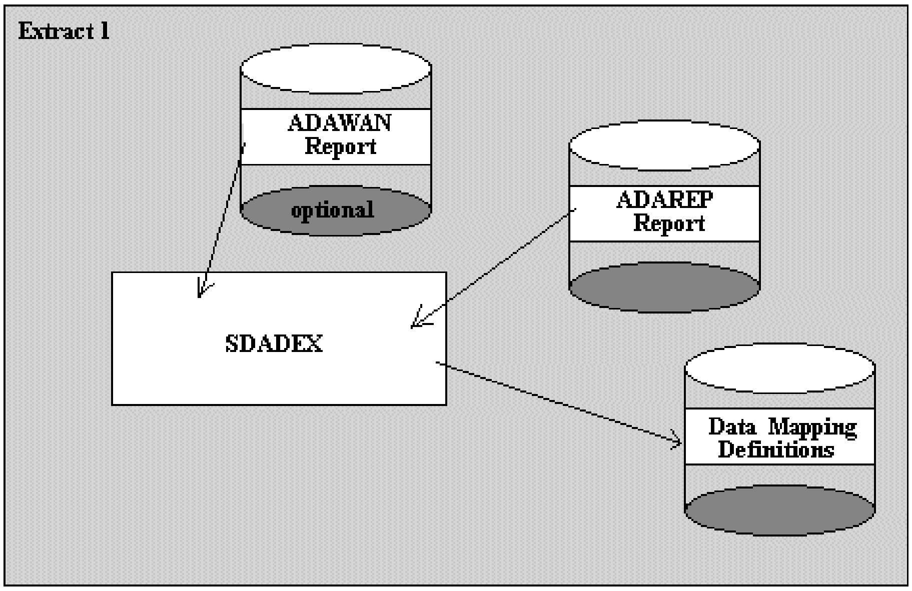


Figure 14–4. Extract Utility 1: SDADEX

Note:
If the ADAREP report is created with no specific file specification, then all of the data mapping definitions for the given database will be generated. Multiple (optional) ADAWAN datasets can be concatenated for input to the utility in this case.

Input for SDADEX

The optional input DDNAME `'SYSIN'` can be used to control SDADEX execution and to allow overrides to the generated definitions.

SUBSYS = xxxx

Here, `xxxx` is the four character name that identifies the ADABAS router name assignment. If not specified, the default name will be ADAB.

DE_SEARCH_ONLY

This parameter causes the utility to generate control definitions that allow the client to only use WHERE columns that are ADABAS descriptors, such as, superde, subde, and hyperde.

MAP_PREFIX = xxx

This is a three character prefix which is appended to the five character file number. It is used as the member name in the generation of the MAP_NAME entry. SDADDM uses this name as the member name of the mapping dataset.

MAX_MU = nnnnn

This is the maximum allowed MU columns generated. The default is 191, if not specified.

This override option is only valid when you extract using an ADAREP report. When you extract using an ADAWAN report, the number of MU occurrence is obtained from the ADAWAN report.

MAX_PE = nnnnn

This is the maximum allowed PE columns generated. The default is 91, if not specified.

This override option is only valid when you extract using an ADAREP report. When you extract using an ADAWAN report, the number of PE occurrence is obtained from the ADAWAN report

BEGIN_OVERRIDES

This card indicates the beginning of the field format/length overrides.

END_OVERRIDES

This card indicates the end of the field format/length overrides.

FILE = nnnnn

This is the file number to be overridden.

FIELD = xx

This is the two character ADABAS field name to be overridden.

FORMAT = x

This is the 1 byte format type to be overridden. The rules of overriding a field format must be acceptable to the rules of data type conversions that ADABAS permits; otherwise, an ADABAS response code may be given due to conversion mismatch.

LENGTH = nnn

This is the (optional) length override.

SCALE = nn

nn equals the number of decimal places for a packed field. This override is only valid when the format of the field is `FORMAT=P`.

CONVERT_U_2_P

This informs the extract to convert all unpacked format fields to packed format.

CONVERT_B_2_I

This informs the extract to convert all 4-byte and 2-byte binary fields to integer and short integer formats.

PE_MU_COUNT

This causes the extract to generate a count field for all MU and PE fields. The generated name is the PE or MU field name plus an underscore followed by the letter "C". For example, if the PE or MU name is ACCOUNTS, the generated name will be ACCOUNTS_C.

EXAMPLE:

In this example, the client only uses ADABAS descriptors for searches. The ADABAS subsystem name is ADAC and has two files.

- FILE 1 has field `AH` and `AR` defined as `NATURAL` dates. AR's length will be overridden to 4 bytes.
- FILE 2 has field `AB`. It will have a character format override.
- FILE 2 contains a packed field `AT`. Two decimal places are required.

```
//SYSIN DD *
DE_SEARCH_ONLY
SUBSYS = ADAC
BEGIN_OVERRIDES
   FILE = 1, FIELD = AH, FORMAT = D
   FILE = 1, FIELD = AR, FORMAT = D, LENGTH = 4
   FILE = 2, FIELD = AB, FORMAT = A
   FILE = 2, FIELD = AT, SCALE = 2
END_OVERRIDES
MAP_PREFIX = PRD
MAX_MU = 3
MAX_PE = 2
```

Output of SDADEX Using an ADAREP Report for the Sample Employee File

If you use the following control card to extract data:

```
//DDDRUCK DD DSN=CSD.AI38.ADA100.ADAREP,DISP=SHR
//SYSIN DD *
BEGIN_OVERRIDES
   FILE = 1, FIELD = AH, FORMAT = D
   FILE = 1, FIELD = AR, FORMAT = D, LENGTH = 4
END_OVERRIDES
MAP_PREFIX = PRD
MAX_MU = 3
MAX_PE = 2
```

you will get the following output:

```
BEGIN TABLE DEFINITION
   DATABASE_NAME=NEON-DB
   ADABAS_DBID=00100
   ADABAS_FILE_NUMBER=00001
   FILE_NAME=EMPLOYEES
   SUBSYSTEM_NAME=ADAB
   MAP_NAME=PRD00001
   FIELD=01,AA,008,A                  AA                 AA
   FIELD=02,AC,020,A                  AC                 AC
   FIELD=01,AE,020,A                  AE                 AE
   FIELD=02,AD,020,A                  AD                 AD
   FIELD=02,AF,001,A                  AF                 AF
   FIELD=02,AG,001,A                  AG                 AG
   FIELD=01,AH,006,D                  AH                 AH
   FIELD=02,AI001,020,A               AI001              AI001
   FIELD=02,AI002,020,A               AI002              AI002
   FIELD=02,AI003,020,A               AI003              AI003
   FIELD=01,AJ,020,A                  AJ                 AJ
   FIELD=02,AK,010,A                  AK                 AK
   FIELD=02,AL,003,A                  AL                 AL
   FIELD=02,AN,006,A                  AN                 AN
   FIELD=02,AM,015,A                  AM                 AM
   FIELD=01,AO,006,A                  AO                 AO
   FIELD=01,AP,025,A                  AP                 AP
   FIELD=02,AR01,004,D                AR01               AR01
   FIELD=02,AR02,004,D                AR02               AR02
   FIELD=02,AS01,005,P                AS01               AS01
   FIELD=02,AS02,005,P                AS02               AS02
   FIELD=02,AT01(001),005,P           AT01001            AT01001
   FIELD=02,AT01(002),005,P           AT01002            AT01002
   FIELD=02,AT01(003),005,P           AT01003            AT01003
   FIELD=02,AT02(001),005,P           AT02001            AT02001
   FIELD=02,AT02(002),005,P           AT02002            AT02002
   FIELD=02,AT02(003),005,P           AT02003            AT02003
   FIELD=02,AU,002,U                  AU                 AU
   FIELD=02,AV,002,U                  AV                 AV
   FIELD=02,AX01,006,U                AX01               AX01
   FIELD=02,AX02,006,U                AX02               AX02
   FIELD=02,AY01,006,U                AY01               AY01
   FIELD=02,AY02,006,U                AY02               AY02
   FIELD=01,AZ001,003,A               AZ001              AZ001
   FIELD=01,AZ002,003,A               AZ002              AZ002
   FIELD=01,AZ003,003,A               AZ003              AZ003
   FIELD=04,H1,004,B                  H1                 H1
   FIELD=06,AU,001,002
   FIELD=06,AV,001,002
   FIELD=04,S1,004,A                  S1                 S1
   FIELD=06,AO,001,004
   FIELD=04,S2,026,A                  S2                 S2
   FIELD=06,AO,001,006
   FIELD=06,AE,001,020
   FIELD=04,S3,012,A                  S3                 S3
   FIELD=06,AR,001,003
   FIELD=06,AS,001,009
END
```

Output of SDADEX - Using Both ADAREP and ADAWAN Reports for the Sample Employee File

If you use the following control card to extract data:

```
//DDDRUCK  DD DSN=CSD.AI38.ADA100.ADAREP,DISP=SHR
//ADAWAN   DD DSN=CSD.AI38.ADA100.ADAWAN,DISP=SHR
//SYSIN DD *
BEGIN_OVERRIDES
   FILE = 1, FIELD = AH, FORMAT = D
   FILE = 1, FIELD = AR, FORMAT = D, LENGTH = 4
END_OVERRIDES
MAP_PREFIX = PRD
MAX_MU = 3
MAX_PE = 2
//
```

and this ADAWAN report:

```
ADACMP COMPRESS                                                    00000100
ADACMP FILE=1                                                      00000200
ADACMP MINISN=1                                                    00000300
ADACMP DEVICE=3380                                                 00000400
ADACMP FNDEF='01,AA,08,A'        EMPLOYEE-ID                       00000500
ADACMP FNDEF='01,AC,20,A'        FIRST-NAME                        00000600
ADACMP FNDEF='01,AE,20,A'        LAST-NAME                         00000700
ADACMP FNDEF='01,AD,20,A'        AALL-DLY-ORIG-ALLOC-SVL-GRP 00000800
ADACMP FNDEF='01,AI,9,U,MU(2)'   HERE-IS-A-SHORT-MU                00000900
ADACMP FNDEF='01,AT,9,U,MU(10)'  HERE-IS-AN-MU                     00000910
ADACMP FNDEF='01,AQ,PE(20)'      HERE-IS-A-PE                      00001000
ADACMP FNDEF='02,AR,6,U'         HERE-IS-1-PE-FIELD                00001100
ADACMP FNDEF='02,AS,6,U'         HERE-IS-2-PE-FIELD                00001200
ADACMP FNDEF='02,AT,6,U'         HERE-IS-3-PE-FIELD                00001210
ADACMP FNDEF='01,AZ,03,A'        JUST-ANOTHER-FIELD                00001300
ADACMP SUPDE='01,S1,04,B'        A-BIG-SUPER-DE                    00001400
```

you will get the following output:

```
BEGIN TABLE DEFINITION
   DATABASE_NAME=NEON-DB
   ADABAS_DBID=00100
   ADABAS_FILE_NUMBER=00001
   FILE_NAME=EMPLOYEES
   SUBSYSTEM_NAME=ADAB
   MAP_NAME=PRD00001
   FIELD=01,AA,008,A   EMPLOYEE_ID                    EMPLOYEE_ID
   FIELD=02,AC,020,A   FIRST_NAME                     FIRST_NAME
   FIELD=01,AE,020,A   LAST_NAME                      LAST_NAME
```

```
FIELD=02,AD,020,A   AALL_DLY_ORIG_ALLOC_SVL_GRP    AALL_DLY_ORIG_ALLOC_SVL_GRP
FIELD=02,AF,001,A          AF                      AF
FIELD=02,AG,001,A          AG                      AG
FIELD=01,AH,006,D          AH                      AH
FIELD=02,AI001,020A        HERE_IS_A_SHORT_MU001   HERE_IS_A_SHORT_MU001
FIELD=02,AI002,020A        HERE_IS_A_SHORT_MU002   HERE_IS_A_SHORT_MU002
FIELD=01,AJ,020,A          AJ                      AJ
FIELD=02,AK,010,A          AK                      AK
FIELD=02,AL,003,A          AL                      AL
FIELD=02,AN,006,A          AN                      AN
FIELD=02,AM,015,A          AM                      AM
FIELD=01,AO,006,A          AO                      AO
FIELD=01,AP,025,A          AP                      AP
FIELD=02,AR01,004,D        HERE_IS_1_PE_FIELD01    HERE_IS_1_PE_FIELD01
FIELD=02,AR02,004,D        HERE_IS_1_PE_FIELD02    HERE_IS_1_PE_FIELD02
FIELD=02,AR03,004,D        HERE_IS_1_PE_FIELD03    HERE_IS_1_PE_FIELD03
FIELD=02,AR04,004,D        HERE_IS_1_PE_FIELD04    HERE_IS_1_PE_FIELD04
FIELD=02,AR05,004,D        HERE_IS_1_PE_FIELD05    HERE_IS_1_PE_FIELD05
FIELD=02,AR06,004,D        HERE_IS_1_PE_FIELD06    HERE_IS_1_PE_FIELD06
FIELD=02,AR07,004,D        HERE_IS_1_PE_FIELD07    HERE_IS_1_PE_FIELD07
FIELD=02,AR08,004,D        HERE_IS_1_PE_FIELD08    HERE_IS_1_PE_FIELD08
FIELD=02,AR09,004,D        HERE_IS_1_PE_FIELD09    HERE_IS_1_PE_FIELD09
FIELD=02,AR10,004,D        HERE_IS_1_PE_FIELD10    HERE_IS_1_PE_FIELD10
FIELD=02,AR11,004,D        HERE_IS_1_PE_FIELD11    HERE_IS_1_PE_FIELD11
FIELD=02,AR12,004,D        HERE_IS_1_PE_FIELD12    HERE_IS_1_PE_FIELD12
FIELD=02,AR13,004,D        HERE_IS_1_PE_FIELD13    HERE_IS_1_PE_FIELD13
FIELD=02,AR14,004,D        HERE_IS_1_PE_FIELD14    HERE_IS_1_PE_FIELD14
FIELD=02,AR15,004,D        HERE_IS_1_PE_FIELD15    HERE_IS_1_PE_FIELD15
FIELD=02,AR16,004,D        HERE_IS_1_PE_FIELD16    HERE_IS_1_PE_FIELD16
FIELD=02,AR17,004,D        HERE_IS_1_PE_FIELD17    HERE_IS_1_PE_FIELD17
FIELD=02,AR18,004,D        HERE_IS_1_PE_FIELD18    HERE_IS_1_PE_FIELD18
FIELD=02,AR19,004,D        HERE_IS_1_PE_FIELD19    HERE_IS_1_PE_FIELD19
FIELD=02,AR20,004,D        HERE_IS_1_PE_FIELD20    HERE_IS_1_PE_FIELD20
FIELD=02,AS01,005,P        HERE_IS_2_PE_FIELD01    HERE_IS_2_PE_FIELD01
FIELD=02,AS02,005,P        HERE_IS_2_PE_FIELD02    HERE_IS_2_PE_FIELD02
FIELD=02,AS03,005,P        HERE_IS_2_PE_FIELD03    HERE_IS_2_PE_FIELD03
FIELD=02,AS04,005,P        HERE_IS_2_PE_FIELD04    HERE_IS_2_PE_FIELD04
FIELD=02,AS05,005,P        HERE_IS_2_PE_FIELD05    HERE_IS_2_PE_FIELD05
FIELD=02,AS06,005,P        HERE_IS_2_PE_FIELD06    HERE_IS_2_PE_FIELD06
FIELD=02,AS07,005,P        HERE_IS_2_PE_FIELD07    HERE_IS_2_PE_FIELD07
FIELD=02,AS08,005,P        HERE_IS_2_PE_FIELD08    HERE_IS_2_PE_FIELD08
FIELD=02,AS09,005,P        HERE_IS_2_PE_FIELD09    HERE_IS_2_PE_FIELD09
FIELD=02,AS10,005,P        HERE_IS_2_PE_FIELD10    HERE_IS_2_PE_FIELD10
FIELD=02,AS11,005,P        HERE_IS_2_PE_FIELD11    HERE_IS_2_PE_FIELD11
FIELD=02,AS12,005,P        HERE_IS_2_PE_FIELD12    HERE_IS_2_PE_FIELD12
FIELD=02,AS13,005,P        HERE_IS_2_PE_FIELD13    HERE_IS_2_PE_FIELD13
FIELD=02,AS14,005,P        HERE_IS_2_PE_FIELD14    HERE_IS_2_PE_FIELD14
FIELD=02,AS15,005,P        HERE_IS_2_PE_FIELD15    HERE_IS_2_PE_FIELD15
FIELD=02,AS16,005,P        HERE_IS_2_PE_FIELD16    HERE_IS_2_PE_FIELD16
FIELD=02,AS17,005,P        HERE_IS_2_PE_FIELD17    HERE_IS_2_PE_FIELD17
FIELD=02,AS18,005,P        HERE_IS_2_PE_FIELD18    HERE_IS_2_PE_FIELD18
FIELD=02,AS19,005,P        HERE_IS_2_PE_FIELD19    HERE_IS_2_PE_FIELD19
FIELD=02,AS20,005,P        HERE_IS_2_PE_FIELD20    HERE_IS_2_PE_FIELD20
FIELD=02,AT01(001),005,P   HERE_IS_AN_MU01001      HERE_IS_AN_MU01001
```

```
   FIELD=02,AT01(002),005,P   HERE_IS_AN_MU01002      HERE_IS_AN_MU01002
   FIELD=02,AT01(003),005,P   HERE_IS_AN_MU01003      HERE_IS_AN_MU01003
   FIELD=02,AT01(004),005,P   HERE_IS_AN_MU01004      HERE_IS_AN_MU01004
   FIELD=02,AT01(005),005,P   HERE_IS_AN_MU01005      HERE_IS_AN_MU01005
   FIELD=02,AT01(006),005,P   HERE_IS_AN_MU01006      HERE_IS_AN_MU01006
   FIELD=02,AT01(007),005,P   HERE_IS_AN_MU01007      HERE_IS_AN_MU01007
   FIELD=02,AT01(008),005,P   HERE_IS_AN_MU01008      HERE_IS_AN_MU01008
   FIELD=02,AT01(009),005,P   HERE_IS_AN_MU01009      HERE_IS_AN_MU01009
   FIELD=02,AT01(010),005,P   HERE_IS_AN_MU01010      HERE_IS_AN_MU01010
   FIELD=02,AT02(001),005,P   HERE_IS_AN_MU02001      HERE_IS_AN_MU02001
   FIELD=02,AT02(002),005,P   HERE_IS_AN_MU02002      HERE_IS_AN_MU02002
   FIELD=02,AT02(003),005,P   HERE_IS_AN_MU02003      HERE_IS_AN_MU02003
   FIELD=02,AT02(004),005,P   HERE_IS_AN_MU02004      HERE_IS_AN_MU02004
   FIELD=02,AT02(005),005,P   HERE_IS_AN_MU02005      HERE_IS_AN_MU02005
   FIELD=02,AT02(006),005,P   HERE_IS_AN_MU02006      HERE_IS_AN_MU02006
   FIELD=02,AT02(007),005,P   HERE_IS_AN_MU02007      HERE_IS_AN_MU02007
   FIELD=02,AT02(008),005,P   HERE_IS_AN_MU02008      HERE_IS_AN_MU02008
   FIELD=02,AT02(009),005,P   HERE_IS_AN_MU02009      HERE_IS_AN_MU02009
   FIELD=02,AT02(010),005,P   HERE_IS_AN_MU02010      HERE_IS_AN_MU02010
   FIELD=02,AU,002,U          AU                      AU
   FIELD=02,AV,002,U          AV                      AV
   FIELD=02,AX01,006,U        AX01                    AX01
   FIELD=02,AX02,006,U        AX02                    AX02
   FIELD=02,AY01,006,U        AY01                    AY01
   FIELD=02,AY02,006,U        AY02                    AY02
   FIELD=01,AZ001,003,A       JUST_ANOTHER_FIELD001   JUST_ANOTHER_FIELD001
   FIELD=01,AZ002,003,A       JUST_ANOTHER_FIELD002   JUST_ANOTHER_FIELD002
   FIELD=01,AZ003,003,A       JUST_ANOTHER_FIELD003   JUST_ANOTHER_FIELD003
   FIELD=01,AZ004,003,A       JUST_ANOTHER_FIELD004   JUST_ANOTHER_FIELD004
   FIELD=01,AZ005,003,A       JUST_ANOTHER_FIELD005   JUST_ANOTHER_FIELD005
   FIELD=01,AZ006,003,A       JUST_ANOTHER_FIELD006   JUST_ANOTHER_FIELD006
   FIELD=01,AZ007,003,A       JUST_ANOTHER_FIELD007   JUST_ANOTHER_FIELD007
   FIELD=01,AZ008,003,A       JUST_ANOTHER_FIELD008   JUST_ANOTHER_FIELD008
   FIELD=01,AZ009,003,A       JUST_ANOTHER_FIELD009   JUST_ANOTHER_FIELD009
   FIELD=01,AZ010,003,A       JUST_ANOTHER_FIELD010   JUST_ANOTHER_FIELD010
   FIELD=04,H1,004,B          H1                      H1
   FIELD=06,AU,001,002
   FIELD=06,AV,001,002
   FIELD=04,S1,004,A          S1                      S1
   FIELD=06,AO,001,004
   FIELD=04,S2,026,A          S2                      S2
   FIELD=06,AO,001,006
   FIELD=06,AE,001,020
   FIELD=04,S3,012,A          S3                      S3
   FIELD=06,AR,001,003
   FIELD=06,AS,001,009
END
```

Table Definition Syntax

DATABASE_NAME=XXXXXX
Name of the database as derived from ADAREP report.

ADABAS_DBID=nnnnn
ID number of database as derived from ADAREP report.

ADABAS_FILE_NUMBER=nnnnn
File number associated with file

file_name=xxxxxxxxxx
Externalized map name used in the SQL syntax to define requested table.

MAP_NAME=XXXXXXXX
A one to eight character internal name describing the map name and member name in the data mapping dataset. This name is generated using the three character map prefix and the five character file number. See MAP_PREFIX=xxx for more information.

subsystem_name=xxxx
The name of the ADABAS subsystem on the host MVS system. The ADABAS subsystem name is assigned to the ADABAS router (SVC) during ADABAS installation time.

Hyperde can be added into the resultant extract dataset manually by replicating entries similar to the type 4 entries. Hyperdes must be defined manually because the fields that comprise the hyperde are known only to the hyperexit and are not reflected in the ADAREP at the time of extract.

EXAMPLE:

For fields where nn=01 through 04, the definition syntax is:

```
FIELD=nn,xx,lll,f,s     column_name       field_name
```

For fields where `nn = 06`, the definition syntax is:

```
FIELD=06,xx,bbb,eee
```

where:

nn=01	Display or selection criteria
nn=02	Display only
nn=03	Selection only
nn=04	Super/Sub/Phonetic descriptor
nn=06	Super field element description

xx ADABAS field name. If the field is an MU, a field, or a field within a PE group, the appropriate indexes must be specified. The generation utility will create the first occurrence of any field of and MU, PE or MU within a PE.

For `field=06`, this is the ADABAS “parent” field name.

lll The length of the data item as viewed by the client application. The length must be consistent with ADABAS allowances for any given data type.

`f` Format of the data as expected by the client. The format must be consistent with the formats allowed by ADABAS for conversion.

In addition to the formats allowed by ADABAS, Shadow also allows the following:

- **D** format indicating that this is a NATURAL date to be returned to the client in ODBC format. The field must be the length of 4 (representing the number of bytes to contain a `P4` field in ADABAS).
- **T** format indicating that this is a NATURAL time to be returned to the client in DBC format. The field must be the length of 7 (representing the number of bytes (representing the number of bytes to contain a P7 field in ADABAS).
- **I** format to allow a field defined as B4 in ADABAS to be returned to the client as SQL_INTEGER. Note that if you try to use the **I** format for a field that is not a B4 format, the SDADDM utility will issue an error and stop
- **J** format to allow a field defined as B2 in ADABAS to be returned to the client as SQL_SMALLINT. Note that if you try to use the **J** format for a field that is not a B2 format, the SDADDM utility will issue an error and stop

s Status of the field definition. This parameter is optional. The valid value is “`D`” for disabled. This allows the definition to be loaded, but remain non accessible to the client.

column_name
The name used in the column headers when information is returned to the client.

eee Ending byte position within parent field.

bbb Beginning byte position within parent field.

field_name
The long field name as known to the client. It is recommended that the `field_name` and `column_name` remain the same unless the

product is being used in conjunction with ADABAS Native SQL batch applications. These names are used for referencing fields in the SQL statements.

These table and field definitions can be customized for the specific client by changing items such as field and column names, adding or deleting field definitions, etc. This should be done before the next step of the extraction process is performed.

SDADDM

The second utility program is SDADDM. It takes the resulting data mapping statements created by the SDADEX utility and populates the Shadow Server Data Mapping Facility with the ADABAS meta data. After the utility SDADM is executed, one or more maps are created. These maps must be placed in the map dataset allocated by the server. For Shadow Direct, the map dataset is pointed to by ddname `'SDBMAPP'`. For Shadow Web Server, the map dataset is pointed to by ddname `'SWSMAPP'`. Please refer to the appropriate chapter in the *User's Guide* for more information on Shadow Data Mapping Facility.

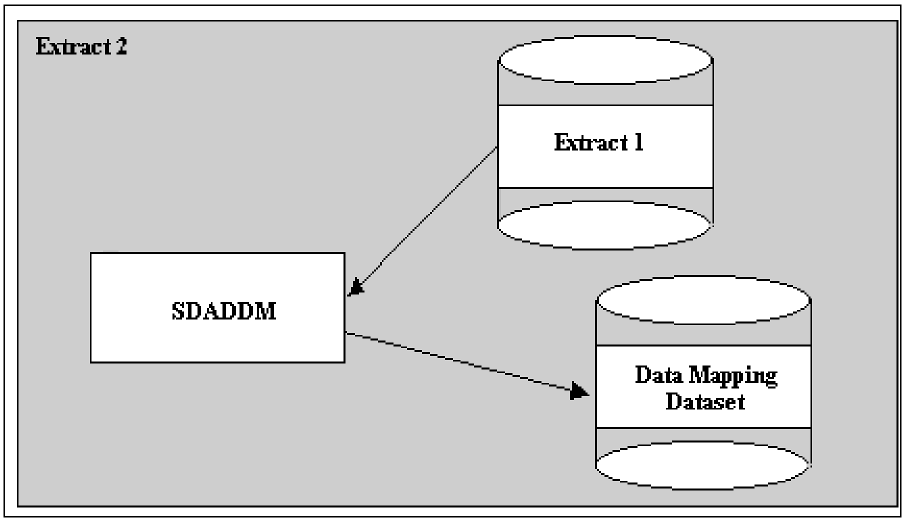


Figure 14–5. Extract Utility 2: SDADDM

JCL can be found in distributed CNTL library as members `'ADABAS1'` and `'ADABAS2'`. `'ADABAS1'` is a JCL example on how use the SDADEX utility, and `'ADABAS2'` is a JCL example on how to use the SDADDM utility.

Dynamic ADABAS Data Mapping

If a data map does not exist in the system, the Shadow ADABAS Server will dynamically build one. The following table name syntax must be included in the client SQL request:

```
ADAx_nnnnn_mmmmm
```

where:

ADAx	The ADABAS subsystem name assigned to the ADABAS Router (SVC).
Nnnnn	The numeric representation of the ADABAS database target (DBID)
Mmmmm	The numeric representation of the ADABAS table (file) number (FNR).

The following example shows how the dynamic ADABAS data mapping concept can be used. It is based on queries executed against the EMPLOYEES file, residing on ADABAS DBID 100 file number 1, on SVC 249 (installed as subsystem ADAB):

Example:

‘select * from ADAB_100_1’
Returns all columns from the table. Since there is no OPTIONS INDEX indicated, only the first occurrence of any PE or MU fields are returned.

‘select * from ADAB_100_1 where AE = “jones” options index = 5’
Returns all columns from the table where the ADABAS field name AE contains the value “`JONES`”. The `OPTIONS INDEX = 5` results in the return of the first through the fifth occurrence of any PE or MU fields.

‘select AE AS1 AS2 from ADAB_100_1 where AE = “jones”’
Returns columns AE, AS1, and AS2 for rows containing “`JONES`” in the AE column.

Cursor Processing

Like Software AG’s Native SQL support, Shadow ADABAS Server supports cursor processing. If the keywords **DECLARE** *`<cursor-name>`* and **CURSOR FOR** are included in the SQL statement, the client application will be able to control the positioning of the cursor by using **OPEN** *`<cursor-name>`*, **FETCH** *`<cursor-name>`* and **CLOSE** *`<cursor-name>`* processing.

Shadow ADABAS Server maintains the required context between client requests to allow single row return on each **FETCH** request.

Example:

```
Call Shadow_ADABAS('declare C001 cursor for select * from employees')
...
Call Shadow_ADABAS('open C001')
...
fetch_loop:
Call Shadow_ADABAS('fetch C001')
check return > 0 exit loop
end_loop;
...
Call Shadow_ADABAS('close C001')
```

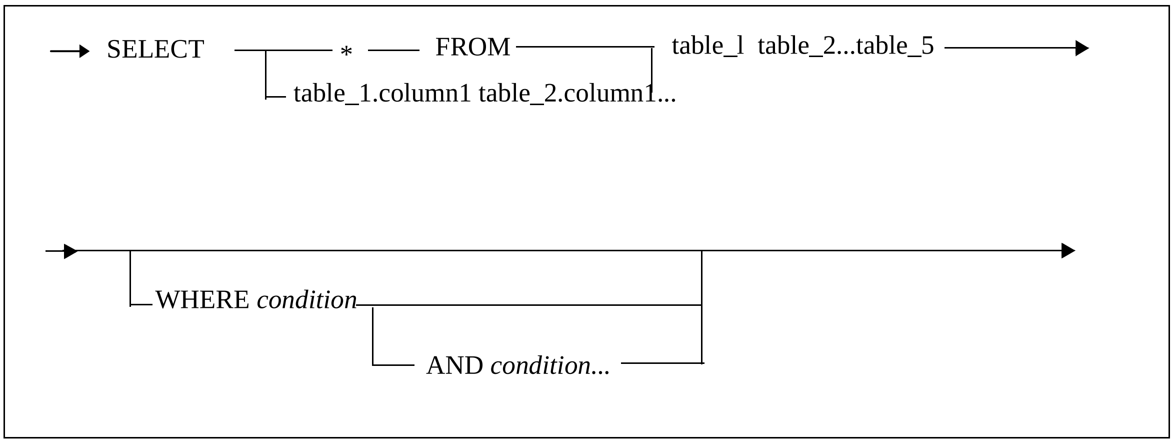
The client application can open as many cursors as required at any given time.

Cursors names must be four characters in length, and it is recommended that alphanumeric characters be used and special character usage be avoided. The cursor name "NEON" is used internally by Shadow products and should not be used in client applications.

Obtaining Data From Multiple ADABAS files

Shadow_ADABAS supports the selection of data from one to five ADABAS tables by allowing a logical joining of the tables through use of common data elements.   Access only is permitted when multiple ADABAS tables are specified in the Select statement. No updating of tables is allowed.

The Select statement requires that each column used in the select be qualified by the ADABAS table nameas shown in the following example:



condition

table_1.column2 *operator*   value
table_l.column2 operator table_2.column2

Shadow_ADABAS transforms the query into multiple nested selects. The order for processing the individual selects is based on table interdependencies within the original query. Conditions that specify values always take precedence.

For example:

```
SELECT * FROM EMPLOYEES, VEHICLES WHERE VEHICLE.PERSONNEL_ID =
EMPLOYEE.PERSONEL_ID AND EMPLOYEE.LAST_NAME = "JONES"
```

will result in the following statement execution:

```
    DECLARE C001 CURSOR FOR
        SELECT * FROM EMPLOYEES WHERE LAST_NAME = "JONES"
    OPEN C001
  ┌─FETCH C001
  │      READ ISN SELECT PERSONELL_ID FROM EMPLOYEES WHERE CURRENT
  │         OF C001
  │      DECLARE C002 CURSOR FOR
  │             SELECT * FROM VEHICLES WHERE VEHICLE_ID = "???????????"
  │      OPEN C002
  │      FETCH C002
  │      LOOP
  └─LOOP
```

For each fetched row from cursor C001, the data returned for the column EMPLOYEE.PERSONNEL_ID is substituted in the C002 select statement. The fetch for C002 is done until all records are exhausted (ADARSP 3) and the outer fetch C001 is then executed to retrieve the next row pointed to by the C001 cursor. The process continues until cursor C001 is exhausted (ADABAS RSP 3).

This logic is similar to the manner in which Natural would handle nested FIND statements.

Each row returned to the client would contain all columns from the EMPLOYEES table as well as all columns from the VEHICLES file.

SQL Syntax Supported

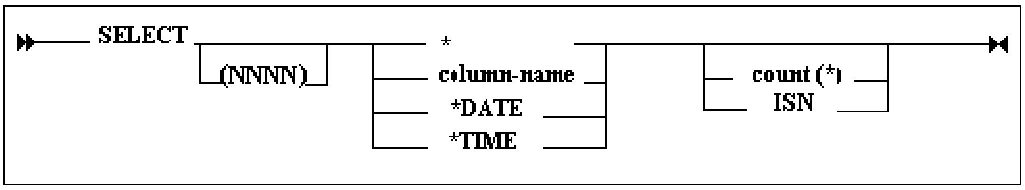
The following list shows all ANSI Standard SQL and Software AG's Native SQL syntax supported by the Shadow ADABAS SQL Server.

Statements	ADABAS Native SQL	ANSI SQL
CLOSE cursor	Y	Y
COMMIT [WORK]	Y	Y
CONNECT	Y	N
DBCLOSE	Y	N
DELETE	Y	Y
FETCH cursor	Y	Y
[FIND] SELECT	Y	Y
HISTOGRAM	N	N
HOLD	Y	N
INSERT	Y	Y
OPEN cursor	Y	Y
READ ISN/LOGICAL/PHYSICAL	Y	N
RELEASE	Y	N
ROLLBACK [WORK]	Y	Y
SET	N	N
SHOW	N	N
TRACE	Y	N
UPDATE	Y	Y

Table 14–1. SQL Syntax Supported

The next section include the syntax diagram for each statement supported by Shadow ADABAS SQL Server.

Selection Criterion



SELECT options are:

SELECT * from xyz
Obtains all enabled columns from table xyz.

SELECT column1 column2 from xyz
Obtains only columns named column1 and column2 from table xyz.

SELECT count(*) from xyz where ...
Returns the number of rows that meet the where criteria. This statement is similar to the NATURAL statement **FIND NUMBER**. If column names are also specified, only columns from the first row that meet the criteria are returned.

SELECT isn from xyz where ...
Returns the ADABAS ISN number assigned to the row that meets the criteria.

SELECT(20) * from xyz where ...
Limits the number of returned rows of data to 20. The number must be in parenthesis immediately following the select keyword. Limits are ignored on statements using cursor processing.

Note:
The keywords count(*) and isn must be coded in the select statement following any column names that are to be returned.

DATE and TIME

The following explains how to obtain the current mainframe date and time and how to set the NATURAL date fields.

Obtaining Mainframe Current Date and/or Time

The current mainframe date and time can be obtained by using the `*DATE` and/or `*TIME` keywords in a **SELECT** statement. Because the current date and time is returned for each and every row returned from the query, you may want to limit the query as shown in this example:

EXAMPLE:

```
SELECT (1) *DATE *TIME FROM table_name
```

Setting NATURAL Date Fields

A NATURAL date field may be indicated to Shadow_ADABAS by using the "`D`" `date_ format` in the Data Mapping Facility input definitions for the SDADDM utility. If this format is used, the client can view or update the specified date column using the ODBC date format. Conversion from ODBC date to a natural date format is done automatically. The length of a "`D`" format type defined in the Data Mapping Facility must be 4.

EXAMPLE:

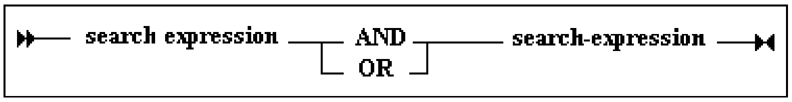
```
UPDATE table_name SET column_name = "1999/01/01"
```

would result in the given date being converted to a NATURAL date format and stored in the requested ADABAS file.

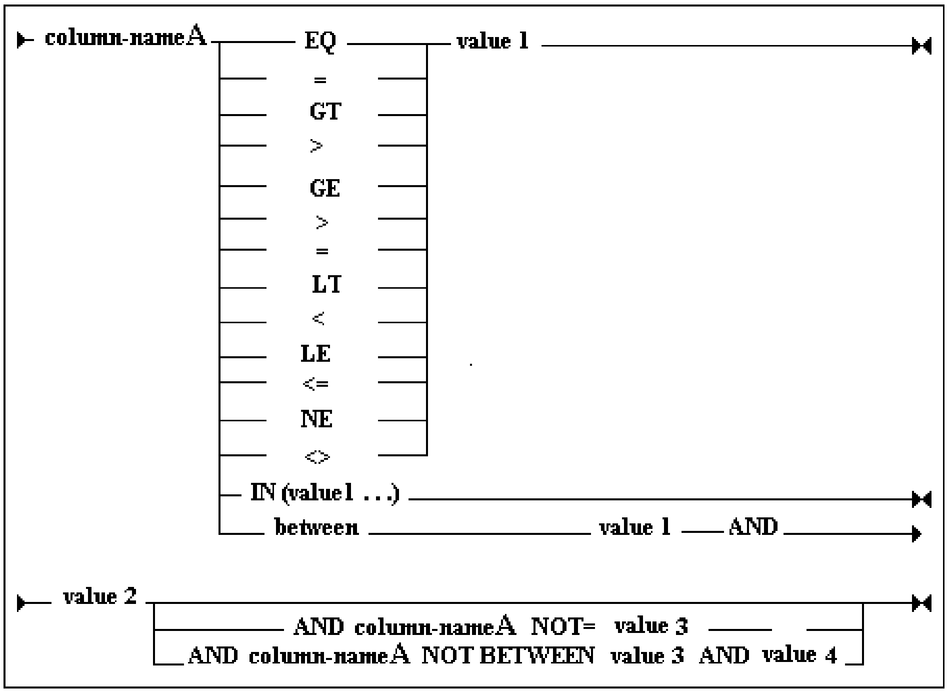
The following sets the specified column_name to the current date:

```
UPDATE table_name SET column_name = *DATE
```

Search Criterion



Search Expression



If you want WHERE criteria to allow only ADABAS descriptor searches, use DE_SEARCH_ONLY during data map creation.

Concatenation of Fields that Comprise a Superde in Search Criteria

When using a superdescriptor that is comprised of multiple parent fields with dissimilar formats, use concatenation notation to create the superdescriptor value.

Note:
There must be the same number of defined type 6 format definitions in the mapping facility as there are field elements that comprise the superdescriptor.

EXAMPLE:

Superdes `S1` is composed of the parent fields `AA`, `AB` and `AC`, where:

- `AA` is defined as A3.
- `AB` is defined as P5.
- `AC` is defined as A2.

To create the superdescriptor in the proper format to send to ADABAS, enter your SQL search criteria as follows:

```
WHERE S1 = "ABC|12345|DE"
```

HEX Notation

Anywhere there is a value required and the value is a non-displaying item, use:

```
X'xx'
```

Where `xx` is a hexidecimal. `xx` must be given in pairs representing the high and low order nibbles of the byte(s).

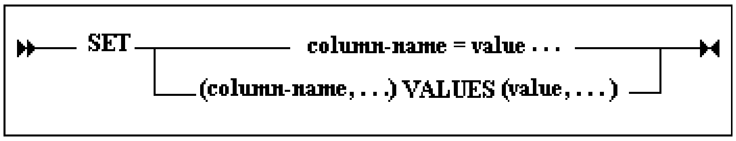
EXAMPLE:

`X'C1'` is the character A.

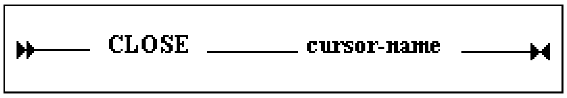
`X'C1F0'` are the characters "A0"

Note:
In most cases, column-names in search criterion need to be ADABAS descriptor fields. The same column name must be specified in all locations designated by column-name A.

ASSIGNMENTS

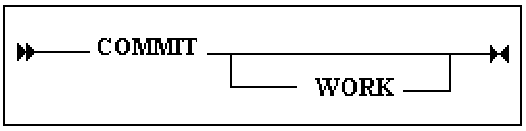

SET
column-name = value . . .
(column-name, . . .) VALUES (value, . . .)

CLOSE



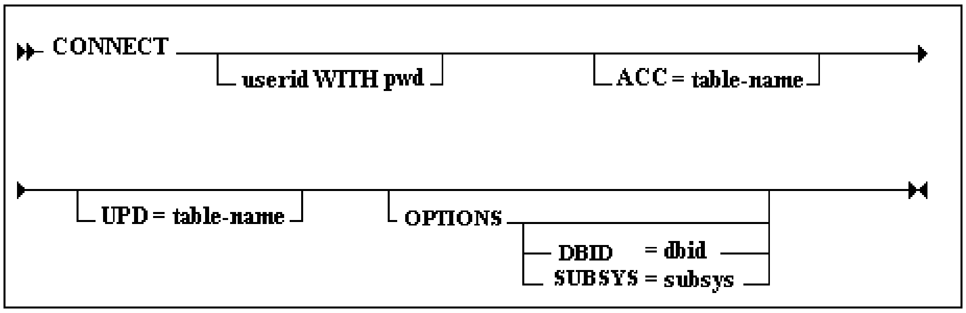
The CLOSE statement allows the termination of a previously opened cursor. Upon termination of the cursor, an ADABAS RC command is issued that uses the four character cursor name as the ADABAS command ID.

COMMIT



The COMMIT statement, which indicates that the application unit or work is to be harded on ADABAS, issues an ADABAS ET command. The COMMIT must be issued by the application program after any UPDATE, DELETE, or INSERT requests. The opposite of the COMMIT statement is the ROLLBACK statement (see ROLLBACK)

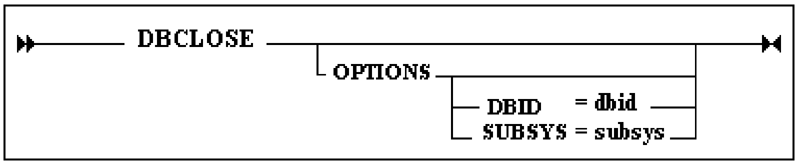
CONNECT



The CONNECT statement allows the application to explicitly establish a connection to ADABAS by issuing an OP command, thereby establishing the ADABAS UQE. If the connect statement is not issued, the ADABAS connection is implicitly established on the first access or update statement issued from Shadow ADABAS Server.

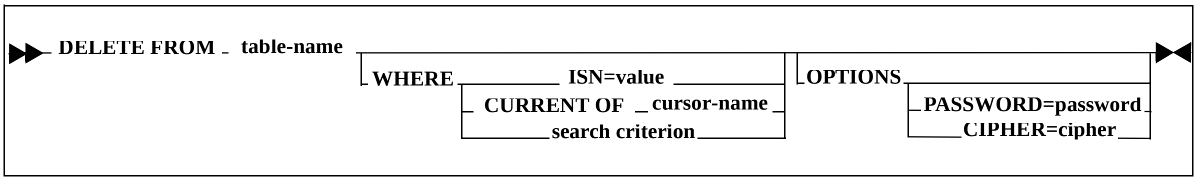
If the ADABAS startup parameter is `OPENRQ=YES`, then the connect statement should be the first Call Shadow_ADABAS statement issued from the client application.

DBCLOSE



The DBCLOSE statement allows the application to explicitly disconnect an ADABAS session without effecting the Shadow connection. The DBCLOSE statement causes an ADABAS CL command to be issued, which also implies an ADABAS COMMIT (ET).

DELETE

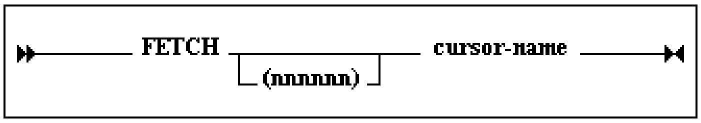


Warning!

Without the WHERE clause, every row in the requested table is deleted.

The DELETE statement allows for the deleting of one or more rows of ADABAS data. Search criteria may be specified as a given ISN value, or the current record pointed to by an open and active cursor. This statement causes an ADABAS E1 command to be issued. It should subsequently be followed by a COMMIT or ROLLBACK command to have the delete committed or rolled back in ADABAS.

FETCH



The FETCH statement allows the retrieval of one or more rows of data from an ADABAS table by issuing the command necessary to continue retrieving rows based on the original statement. If the limiter specification (nnnnnn) is not specified the number of rows returned is 1.

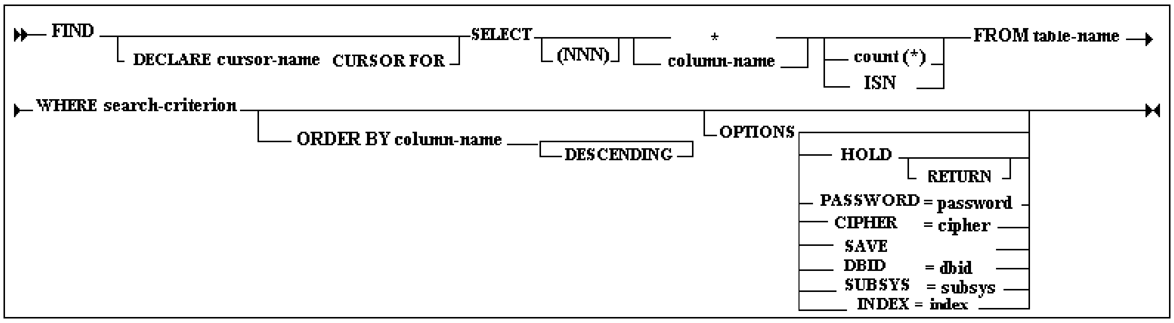
Note:

This command can only be executed after a statement containing a DECLARE CURSOR FOR clause has been executed followed by an OPEN cursor statement. The FETCH command must be followed by a CLOSE cursor statement.

The following pseudo logic describes the use of the fetch statement:

```
DECLARE C001 CURSOR FOR SELECT * FROM EMPLOYEES
OPEN C001
dowhile (SQL_RC = 0)
    FETCH C001
loop
CLOSE C001
```

{FIND} SELECT



The SELECT statement allows the collection of data from ADABAS tables (files). The selection criteria determines which ADABAS columns (fields) are returned, while the where criteria determines the set of rows that are returned to the application. In ADABAS direct call terminology, the selection criteria determines what is generated in the ADABAS Format Buffer and the where criteria determines what is created in the ADABAS Search and Value Buffers. The order by clause executes the appropriate commands for an ADABAS sort.

Options Index is only valid for SELECT statements using dynamic mapping.

Note:
If the FIND keyword is omitted in the statement, Shadow ADABAS Server will determine the best command type for the request based on the contents of the request.

Example:

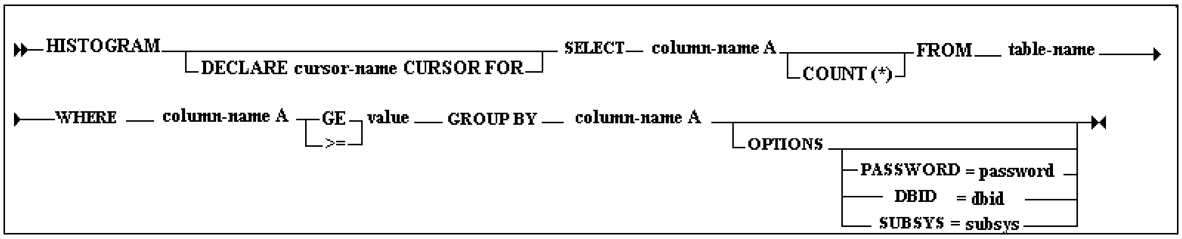
'select * from employees'
Results in an L2/RC command sequence to ADABAS.

'select * from employees where last_name >= "jones"'
Results in an L3/RC command sequence.

'select * from employees where last_name = jones and sex = "f"'
Results in an S1/L1/RC command sequence.

'select count(*) from employees where last_name = "Jones"'
Results in a single S1/RC command sequence. The ISN quantity field of the ADABAS Control block is returned to the client.

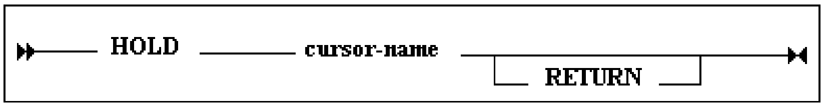
HISTOGRAM



The histogram statement allows the display of key (like descriptor, superdescriptor, or hyperdescriptor) values contained in an ADABAS table. If the `COUNT(*)` keyword is present, a count of each value present in the ADABAS table is returned. The statement generates an ADABAS L9 command sequence.

Note:
All column-names must be the same in this statement and the column name must be an ADABAS descriptor. The same column name must be specified in all locations designated by `column-name A`.

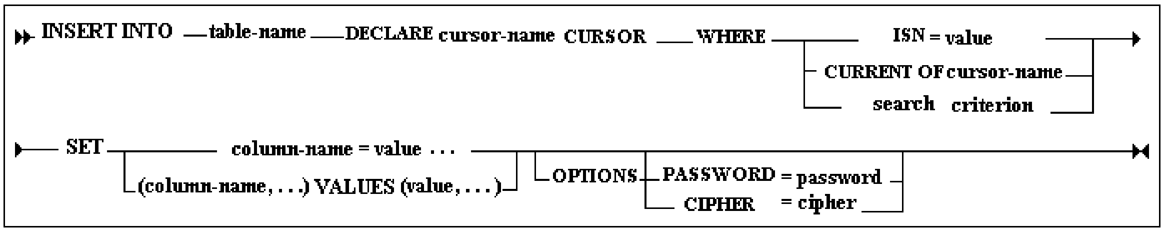
HOLD



The HOLD statement allows for the locking of a single row during cursor processing. This eliminates the need to lock an entire set of records because only the row that the cursor presently represents is locked. An ADABAS HI command is issued for the current ISN in the previous fetch statement.

```
DECLARE C001 CURSOR FOR SELECT * FROM EMPLOYEES
OPEN C001
dowhile (SQL_RC = 0)
   FETCH C001
  if  (row meets condition) then
     HOLD C001
     UPDATE Ö
     COMMIT
  ifend
doend
```

INSERT



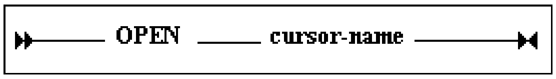
The INSERT statement allows rows of data to be added to an ADABAS table. The ADABAS command issued is N1, unless the where criteria is coded with the ISN= value. In which case, an ADABAS N2 command is issued using the ISN number provided. The set criteria describes the columns and data values that will be placed into the inserted row. The insert statement is generally followed by a COMMIT or ROLLBACK statement.

EXAMPLE:

To add a row to the EMPLOYEES table with a last_name of `"Jones"` and `first_name` of SARAH:

```
INSERT INTO EMPLOYEES SET LAST_NAME = "JONES" FIRST_NAME = "SARAH"
```

OPEN



The OPEN statement is used to indicate the start of cursor processing for a prior given statement. No ADABAS command is issued for this statement. The ADABAS Server creates the internal control structures to maintain the state of the cursor. See the FETCH and CLOSE statements.

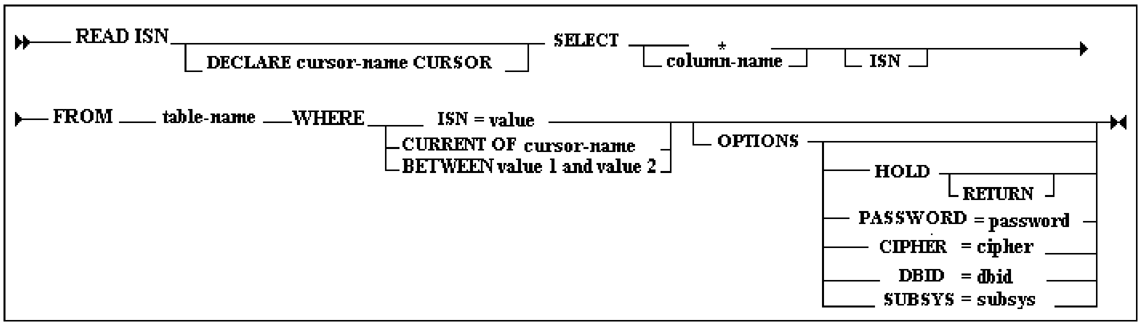
EXAMPLE:

The following pseudo logic describes the use of the open statement:

```
DECLARE C001 CURSOR FOR SELECT * FROM EMPLOYEES
OPEN C001
dowhile (SQL_RC = 0)
    FETCH C001
loop
CLOSE C001
```

READ

READ ISN



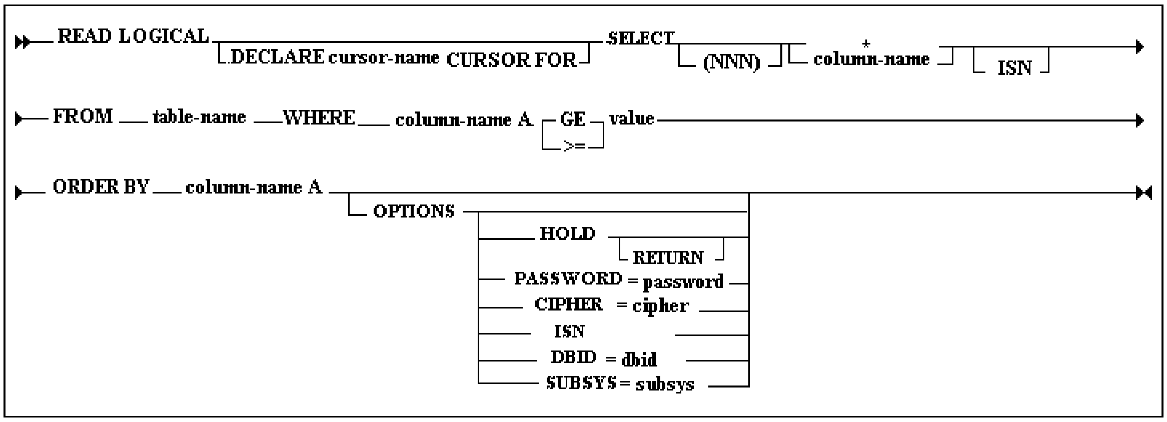
The READ ISN statement allows the application to target a specific row within an ADABAS table using the ADABAS ISN as the search criteria. An ADABAS L1 command is issued for this statement. Because the result set is always returns one row, the READ ISN statement does not allow cursor processing. If the "`BETWEEN`" clause is used, each and every ISN in the range is requested. The ADABAS response 113 (missing ISN) response code is suppressed in this case.

EXAMPLE:

To obtain all columns from ISN 100 in ADABAS table `EMPLOYEES`:

```
READ ISN SELECT * FROM EMPLOYEES WHERE ISN = 100
```

READ LOGICAL



Note:
The same column name must be specified in all locations designated by `column-name A`.

The READ LOGICAL statement allows the application to read ADABAS data logically in the order of a given key (like descriptor or superdescriptor) value. We recommend that cursor processing be used, especially if the result set to be returned is large. This allows you to control the number of rows returned in a single request and the application to control when to terminate the read logical sequence. The read logical generates ADABAS L3 command sequences.

EXAMPLE:

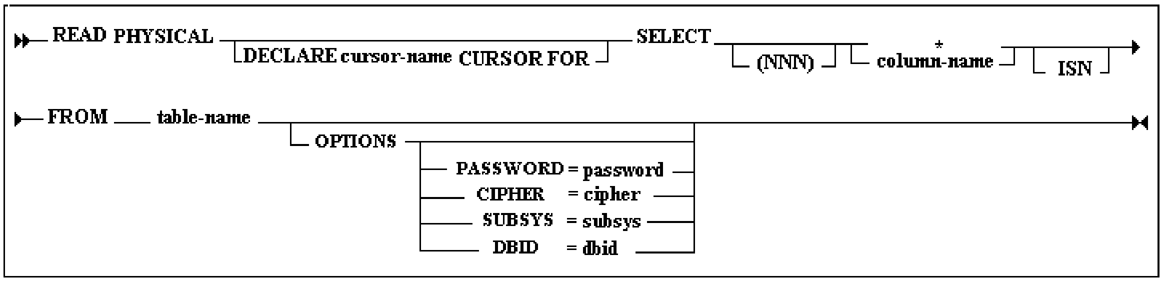
To read all columns from the `EMPLOYEES` table starting from the `LAST_NAME` of `"J"`:

```
READ LOGICAL SELECT * FROM EMPLOYEES WHERE LAST_NAME >= "J"
ORDER BY LAST_NAME
```

If the above query resulted in a large result set, then the application must wait until all the rows of the selection are returned. A more efficient way of coding this selection is:

```
READ LOGICAL DECLARE C001 CURSOR FOR SELECT * FROM EMPLOYEES
WHERE LAST_NAME >= "J" ORDER BY LAST_NAME
OPEN C001
dowhile (SQL_RC = 0)
  FETCH (20) C001
  if  LAST_NAME > "J" then
      leave dowhile
doend
CLOSE C001
```

READ PHYSICAL



The READ PHYSICAL statement allows the reading of ADABAS tables as they are physically stored in the ADABAS table. No key values are used and all data is read. An ADABAS L2 command sequence is generated.

Care must be taken for large ADABAS tables. If the table is large, the response time to the application will be long. We recommend that cursor processing be used whenever wait times need to be minimized.

EXAMPLE:

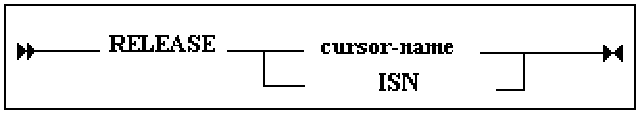
To read all rows, all columns in the EMPLOYEES table:

```
SELECT * FROM EMPLOYEES
```

To minimize the application wait time, use cursor processing:

```
READ PHYSICAL DECLARE C001 CURSOR FOR SELECT * FROM EMPLOYEES
OPEN C001
dowhile (SQL_rc = 0)
FETCH (50) C001
doend
CLOSE C001
```

RELEASE

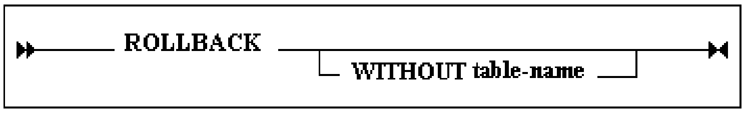


The RELEASE cursor allows for the release of an ADABAS resource. This generates an ADABAS RI command.

EXAMPLE:

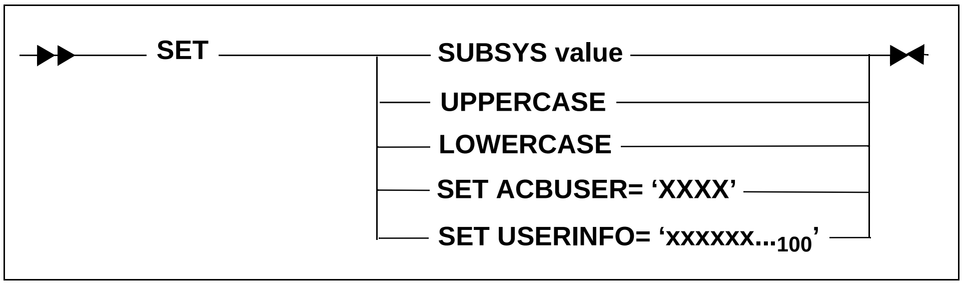
```
RELEASE C001
```

ROLLBACK



The ROLLBACK statement causes ADABAS to back out UPDATEs, INSERTs and/or DELETEs executed after the previous COMMIT or ROLLBACK statement. An ADABAS BT command is issued.

SET



SUBSYS value

If the SUBSYS keyword is used, xxxx will be an ADABAS Subsystem name that corresponds to an ADABAS Router (SVC) number that was assigned during ADABAS installation. This allows the Shadow Data Mapping definition subsystem name to be overridden during execution time.

UPPERCASE and LOWERCASE

These parameters allow the client to tell Shadow ADABAS Server how to translate the data portion of the statements sent to be processed. Lowercase indicates that the data should be left in the mode in which the client has keyed it. If uppercase is used, Shadow ADABAS Server accepts input in either upper or lower case, and then changes all characters to uppercase values. The default is uppercase.

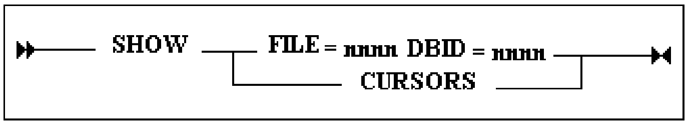
SET ACBUSER = 'XXXX'

Here, xxxx is the user supplied 4 character data that will be placed into the ADABAS Control Block User field (ACBUSER).

SET USERINFO = "xxxxxx...100"

Here, xxxxxx can be from 1 to 100 bytes of user supplied character data. This also checks to see if the ADABAS ADALNK routine has been assembled with at least 100 bytes of userinfo area set. The LNUINFO equate within ADALNK must be set to at least 100 bytes and ADALNK reassembled. This data is passed to the ADABAS user exits as descibed in the *ADABAS DBA Reference Manual*.

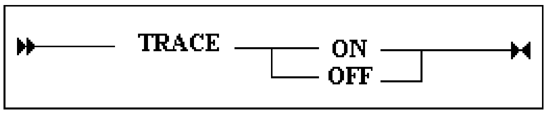
SHOW



If the keywords `FILE` and `DBID` are used, this statement returns the ADABAS FDT for the given ADABAS file and dbid number using an ADABAS LF command.

If cursors is requested, the cursor control blocks are dumped to the client. This information is useful for debugging only in conjunction with NEON's Software Support Group.

TRACE



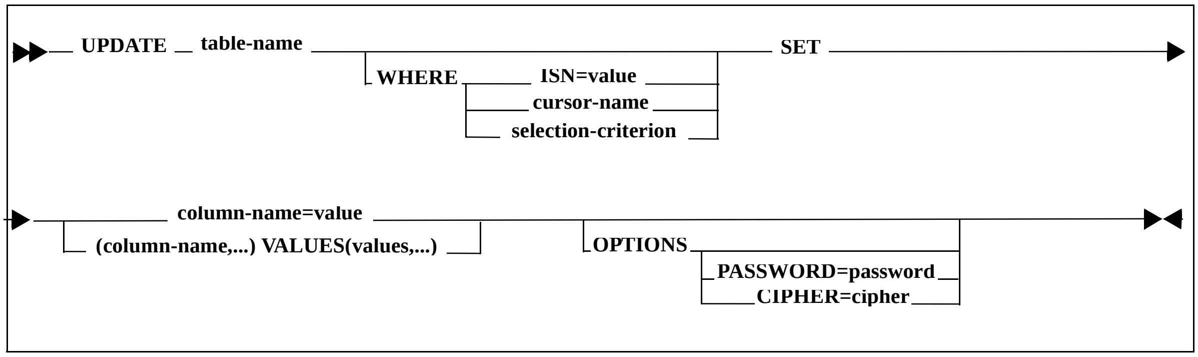
The TRACE statement allows the display of the ADABAS control blocks generated and executed for a given statement. A before and after image of the ADABAS ACB FB RB SB VB are displayed to the application via a column named TRACE. All information is accumulated for all executed statements after the `TRACE ON` is executed. The trace result set is returned for the `TRACE OFF` statement.

EXAMPLE:

To trace the ADABAS buffers created for a `READ ISN` statement:

```
TRACE ON
READ ISN SELECT * FROM EMPLOYEES WHERE ISN = 100
TRACE OFF
```

UPDATE



Warning!
Without the WHERE clause, every row in the requested table is updated.

If the record currently being updated is being used by another user, you will get an ADABAS response code of -143. If a record was partially updated, we recommend that you issue a ROLLBACK.

ADABAS User Identification

The Shadow ADABAS interface uses the sign on user id passed either by the ODBC driver or by the Web browser’s security interface during logon for client authentication. The ODBC client driver and the Shadow Server generate a unique identifier for each application instance. The ADABAS server uses this identifier to establish a connection with ADABAS that allows each application to be viewed as a distinct ADABAS user. This virtual connection identifier allows for the same client to have distinct UQE’s for each application running on the client platform. Commands from one client application do not effect commands issued from another instance of the same application running on the client. The virtual connection ID is a 4 byte binary number.

Loss of Client Connectivity

If client connectivity is lost, the Shadow ADABAS component will issue a ROLLBACK and DBCLOSE on behalf of the client. The ADABAS UQE is eliminated for this connection once the DBCLOSE is issued. Dangling UQEs for application failures are not permitted.

Tracing

Shadow ADABAS provides two types of tracing.

- Centralized tracing is performed on the mainframe using Shadow’s Trace Browse facility. This trace is very useful in determining problems with client requests in a production environment. All SQL statements and resulting ADABAS requests from all clients are written to the Shadow’s Trace Browse for viewing by the Systems Administrator.
- Decentralized or client tracing can be performed during application development using the SQL TRACE ON and TRACE OFF statements. The TRACE ON statement causes the Shadow ADABAS Server to log all ADABAS buffer images, until the application issues a TRACE OFF request. When the TRACE OFF request is processed, the buffered ADABAS buffer images are returned to the client in ODBC format.

The ODBC Administrator and ADABAS Usage

(For Shadow Direct Only) The Shadow ADABAS feature can be used with all supported levels of the Shadow Direct ODBC Driver. For ODBC drivers that do not have the DBMS indicator, the DB2 SUBSYSTEM name is used to indicate an ADABAS connection type. In this case, the DB2 SUBSYSTEM name must be "NONE".

Compatibility with Other Software AG Products

At this time, there are no known incompatibilities with any Software AG products.

Installation Requirements

ADABAS Link Routine

The ADABAS ADALNK routine must be available to Shadow ADABAS Server during execution. You can either copy the routine from the standard ADABAS product load library to the Shadow Server load library, or provide the ADABAS load library in a Shadow STEPLIB concatenation. If Shadow is running as a started task, care must be taken not to lose authorization.

Note:
The ADALNK routine will execute re-entrant. Shadow loads and modifies the ADABAS link routine to run re-enterent. If any user exit is used in your ADALNK routine, it needs to be re-entrant also.

Messages and Codes

Message/Code	Explanation	Action/Cause
-2 thru -255	Specific ADABAS error codes, for example, -9 SQLCODE is an ADABAS response 9. (See Software AG ADABAS Messages and Codes Manual for more information about response codes).	Determine cause of ADABAS response code and resubmit request.
-2001 thru -2999	SQL syntax errors. Last three digits of error number represent keyword or keywords within syntax in error. If digits indicate a number greater than number of words in SQL string, probable cause for syntax error is missing required keyword(s).	Correct syntax error and resubmit request.
-3998	ADABAS link routine not found in appropriate control structures.	Report this internal error to NEON Systems Support group.
-3999	ADABAS Control Block not passed to internal function that calls ADABAS link routine.	Report this internal error to NEON Systems Support group.
-4001	No internal ADABAS user control block found for this request.	Report this internal error to NEON Systems Support group.
-4003	Unknown cursor name used in SQL statement.	Insure cursor name used was used in DECLARE clause of a successfully completed request. Resubmit request after determining proper cursor name.

Message/Code	Explanation	Action/Cause
-4004	Cursor name used in SQL statement already open.	Client attempted to OPEN a cursor that had already been OPENed.
-4005	Cursor name used in SQL statement not open.	Client attempted to FETCH a cursor that had not previously been OPENed.
-4006	ADABAS control block not passed to internal ADABAS interface.	Report this internal error to NEON Systems Support group.
-4007	Client requested an INSERT but no data values were given to be inserted.	Include values to be inserted in INSERT statement.
-4008	Client requested selection criteria that included a COUNT(*) request. No WHERE criteria was given.	Add WHERE criteria to SQL statement and resubmit request.
-4012	No Data Map found for requested file.	Specified map name is either misspelled or not defined in the Data Mapping Facility. If not defined, use Data Mapping Facility extract utilities to define ADABAS file to Data Mapping facility.
-4013	No Data Map column definition found for requested column name.	Specified column is either misspelled or not defined in Data Mapping Facility. If not defined, use Data Mapping Facility extract utilities to define ADABAS file to Data Mapping facility.
-4014	Data Map definition not enabled.	Requested Data Map column is not enabled and cannot be requested in selection criteria and/or where criteria.
-4025	UPDATE, INSERT, DELETE, COMMIT, ROLLBACK or HOLD commands not allowed	Database server for ADABAS was started on READ-ONLY, i.e., READONLY parameter was set to YES.
-4095	License code for Shadow products not allowing ADABAS component to execute.	Contact NEON Systems product distribution for license code that permits use of ADABAS component.
-9000	Subsystem name not defined on operating system.	Insure that four character subsystem name is spelled correctly and defined to the system. Resubmit the request.

CHAPTER 15: Shadow_VSAM and Shadow_VSAM for CICS

This chapter describes the Shadow_VSAM server, which provides a method of accessing existing VSAM data from the desktop. It also covers the Shadow_VSAM component for CICS, which provides a method of accessing and updating CICS assigned KSDS VSAM files from the desktop.

What are they?

The Shadow_VSAM server and Shadow_VSAM for CICS server are new add-on components to NEON's line of Shadow Server products, providing reliable, high-performance access to VSAM data.

Figure 15–1 and Figure 15–2 demonstrate the architectural flexibility of both servers as they provide desktop client and Web Browser access, while extending the life span and improving the investment return of existing VSAM data.

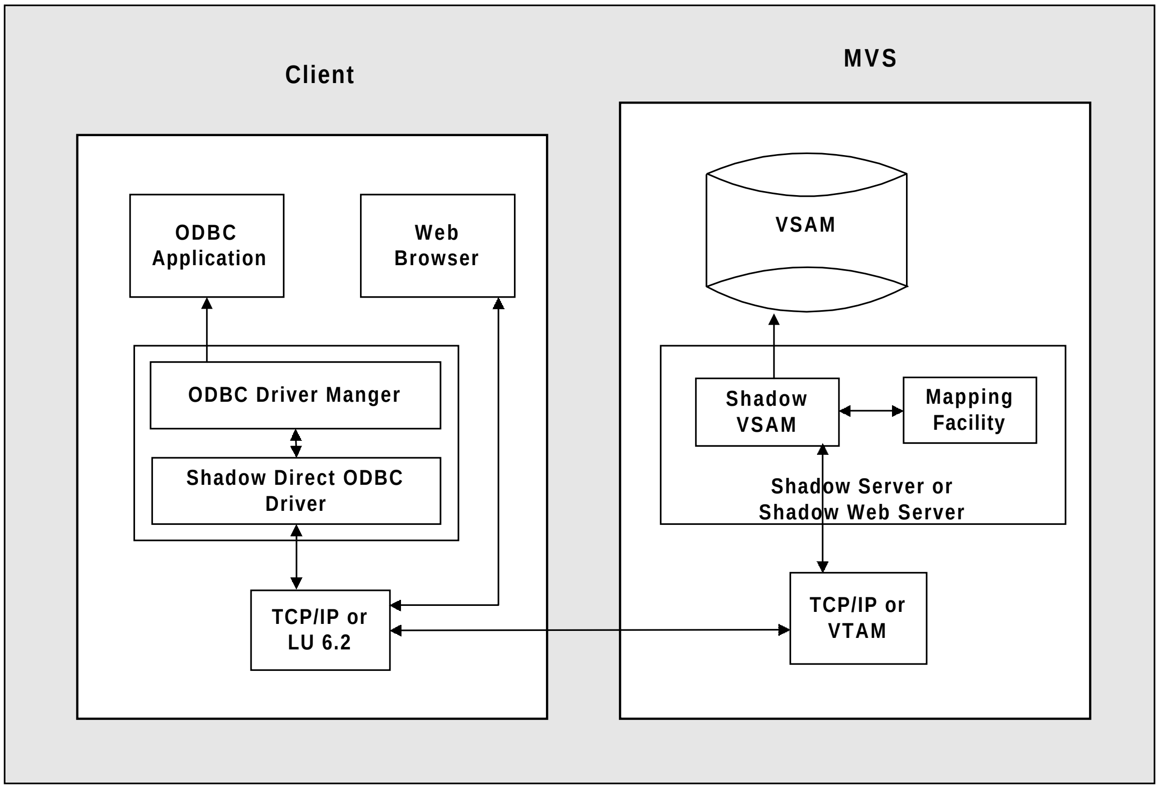


Figure 15–1. Shadow_VSAM Server Environment

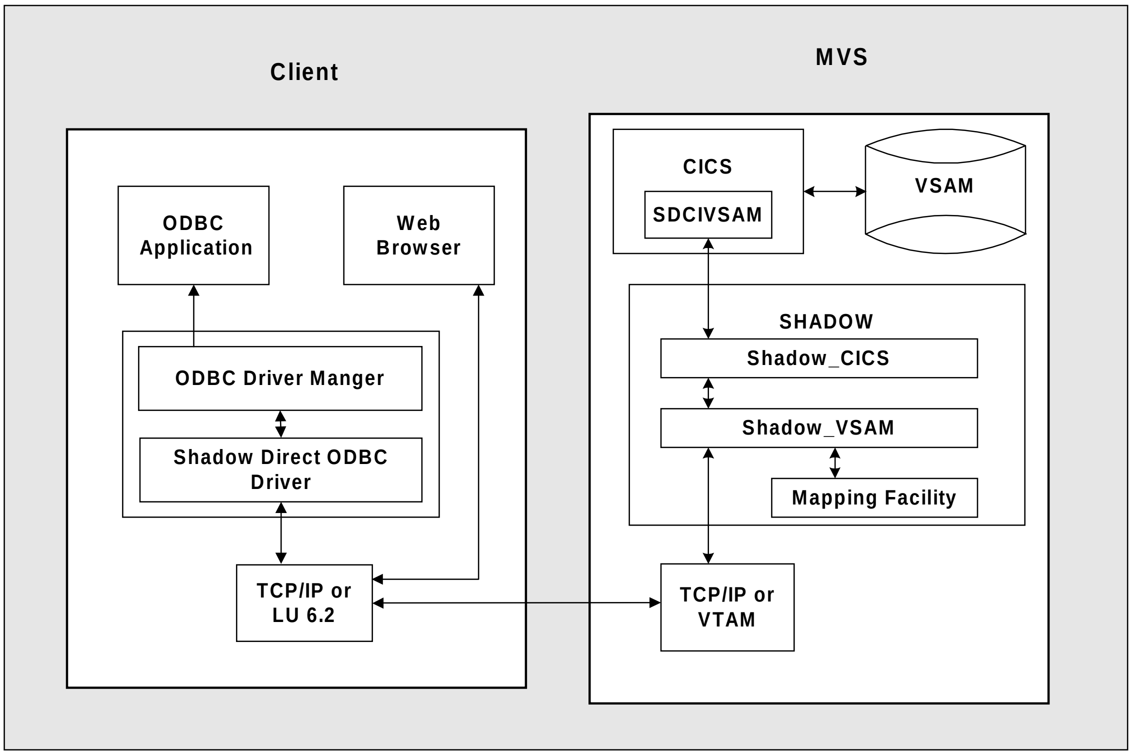


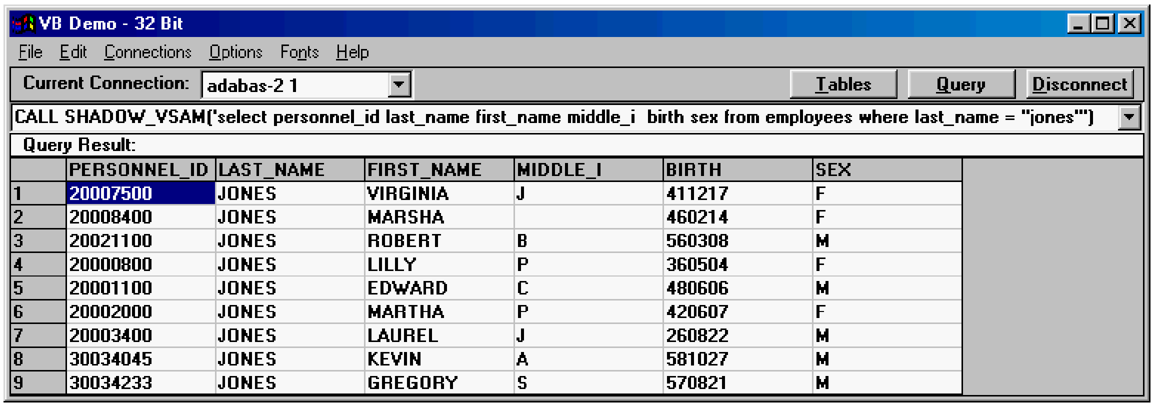
Figure 15–2. Shadow_VSAM for CICS Server Environment

How do they work?

Both Shadow_VSAM and Shadow_VSAM for CICS servers transform a client SQL request into single or multiple VSAM calls. The client requests arrive via the CALL SHADOW_VSAM client interface.

The following example shows a SQL statement requesting the personnel ID number, last name, first name, middle initial, birthdate and sex of all employees whose last name is Jones:

```
CALL SHADOW_VSAM(‘select personnel_id last_name first_name
middle_i birth sex from employees where last_name = “jones”)
```


VB Demo - 32 Bit
File Edit Connections Options Fonts Help
Current Connection: adabas-2 1
Tables
Query
Disconnect
Query Result:

CALL SHADOW_VSAM('select personnel_id last_name first_name middle_i birth sex from employees where last_name = "jones"')

	PERSONNEL_ID	LAST_NAME	FIRST_NAME	MIDDLE_I	BIRTH	SEX
1	20007500	JONES	VIRGINIA	J	411217	F
2	20008400	JONES	MARSHA		460214	F
3	20021100	JONES	ROBERT	B	560308	M
4	20000800	JONES	LILLY	P	360504	F
5	20001100	JONES	EDWARD	C	480606	M
6	20002000	JONES	MARTHA	P	420607	F
7	20003400	JONES	LAUREL	J	260822	M
8	30034045	JONES	KEVIN	A	581027	M
9	30034233	JONES	GREGORY	S	570821	M

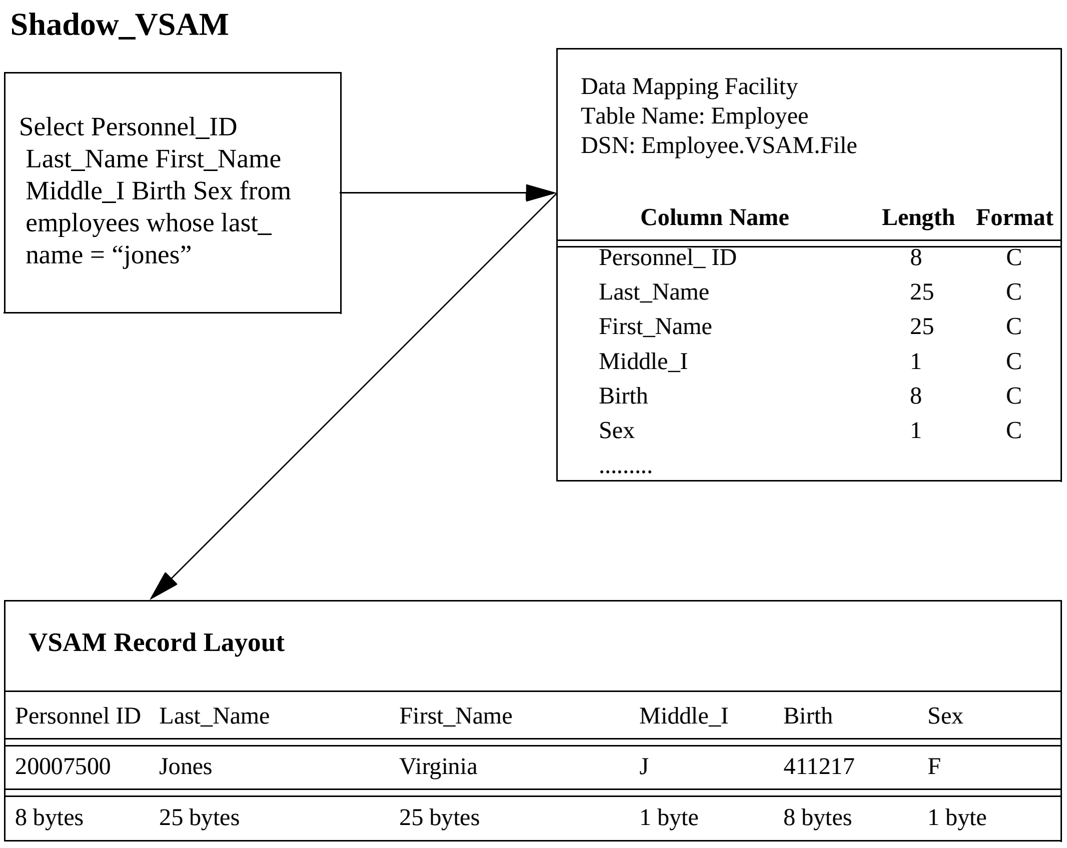
Figure 15–3. SQL Statement Example

Note:

You must include “`CALL SHADOW _VSAM`” wrapper as part of your SQL statement as shown in the example above.

Shadow_VSAM uses the Data Mapping Facility to maintain information that describes the VSAM record layout. Access to the Data Mapping Facility is provided through the Shadow OS/390 Web Server Primary Options Menu.

A VSAM record map definition is created based on the SQL statement processing requirements as shown in the following diagram:


Shadow_VSAM
Select Personnel_ID Last_Name First_Name Middle_I Birth Sex from employees whose last_ name = “jones”
Data Mapping Facility
Table Name: Employee
DSN: Employee.VSAM.File
VSAM Record Layout

Column Name	Length	Format
Personnel_ ID	8	C
Last_Name	25	C
First_Name	25	C
Middle_I	1	C
Birth	8	C
Sex	1	C
....		

Personnel ID	Last_Name	First_Name	Middle_I	Birth	Sex
20007500	Jones	Virginia	J	411217	F
8 bytes	25 bytes	25 bytes	1 byte	8 bytes	1 byte

Figure 15–4. Shadow_VSAM using Data Mapping Facility

Shadow Data Mapping Facility

Before you can use the VSAM server, you must first define the VSAM data file to the Data Mapping Facility.

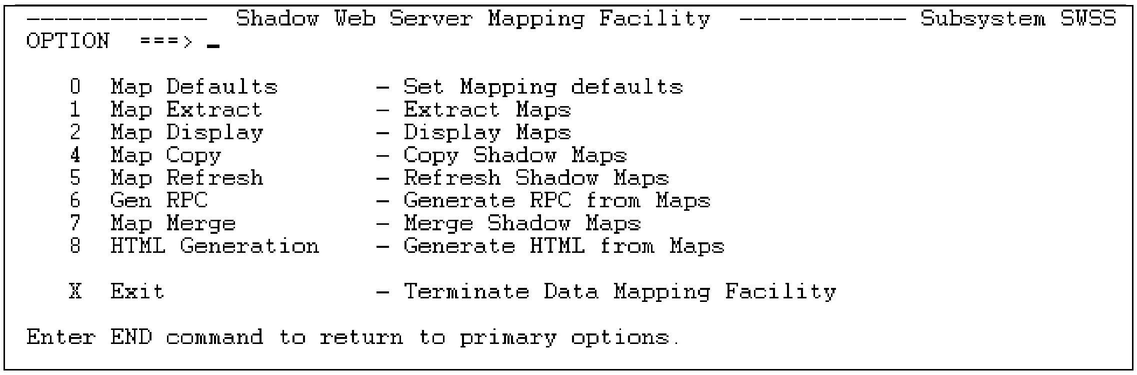
Note:
The VSAM file must be available to Shadow Server at initialization time with the proper VSAM defined share options to allow access to the file.

Defining the VSAM Data Set Files

To define the VSAM data set files to the Data Mapping Facility:

1. Select the Data Mapping Facility from the Shadow Web Server Primary Options Menu.

   The Mapping Facility panel appears.

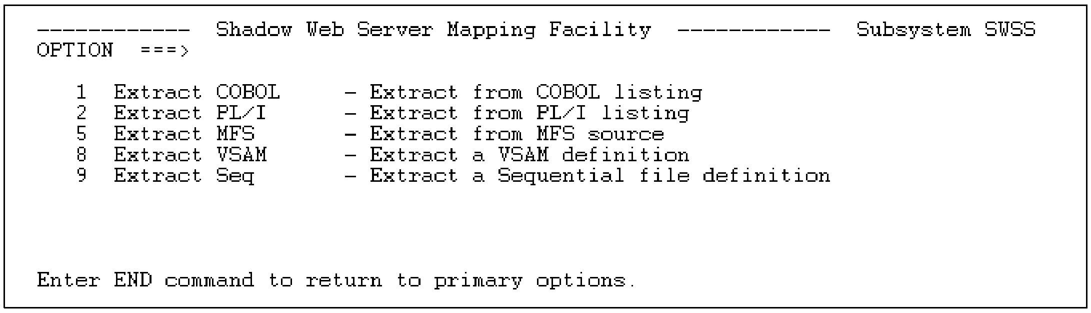


```
-------------  Shadow Web Server Mapping Facility  ------------ Subsystem SWSS
OPTION  ===> _

   0  Map Defaults       - Set Mapping defaults
   1  Map Extract        - Extract Maps
   2  Map Display        - Display Maps
   4  Map Copy           - Copy Shadow Maps
   5  Map Refresh        - Refresh Shadow Maps
   6  Gen RPC            - Generate RPC from Maps
   7  Map Merge          - Merge Shadow Maps
   8  HTML Generation    - Generate HTML from Maps

   X  Exit               - Terminate Data Mapping Facility

Enter END command to return to primary options.
```

Figure 15–5. Data Mapping Facility Options Menu

2. Select the Map Extract option.

   The map Extract Options Menu appears.

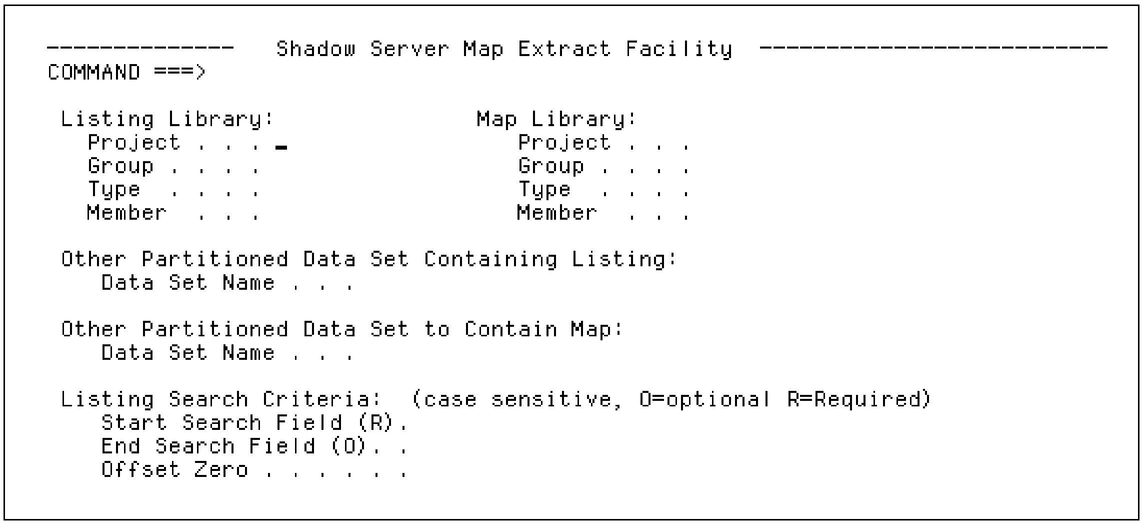


```
------------  Shadow Web Server Mapping Facility  ------------  Subsystem SWSS
OPTION  ===>

   1  Extract COBOL     - Extract from COBOL listing
   2  Extract PL/I      - Extract from PL/I listing
   5  Extract MFS       - Extract from MFS source
   8  Extract VSAM      - Extract a VSAM definition
   9  Extract Seq       - Extract a Sequential file definition



Enter END command to return to primary options.
```

Figure 15–6. Map Extract Options Menu

3. Select Extract VSAM.

   The Shadow Server Map Extract Facility panel appears.



```
--------------   Shadow Server Map Extract Facility  --------------------------
COMMAND ===>

 Listing Library:               Map Library:
   Project . . . _                 Project . . .
   Group . . . .                   Group . . . .
   Type  . . . .                   Type  . . . .
   Member  . . .                   Member  . . .

 Other Partitioned Data Set Containing Listing:
    Data Set Name . . .

 Other Partitioned Data Set to Contain Map:
    Data Set Name . . .

 Listing Search Criteria:  (case sensitive, O=optional R=Required)
    Start Search Field (R).
    End Search Field (O). .
    Offset Zero . . . . . .
```

Figure 15–7. Shadow Server Map Extract Facility

4. Either:

   a. Enter the information for the Listing and Map libraries.
   b. Specify another partitioned dataset for the Listing or Map libraries.

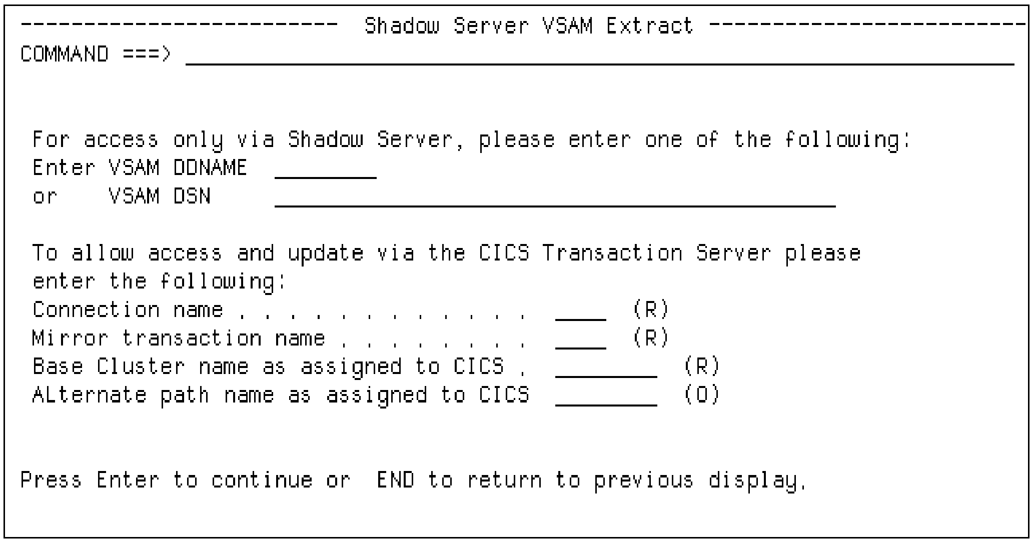
   The Map Library Member name is the name Shadow Server associates with this map.

5. Enter information in the Listing Search Criteria fields.

   This field is used to search the listing dataset for the starting point of the language dependent data declaration. It is recommended that the full qualified name of the declaration be used as it appears in the listing.

6. Press <ENTER>.

   The Shadow Server VSAM Extract panel appears.



```
------------------------  Shadow Server VSAM Extract ------------------------
COMMAND ===> ________________________________________________________________


 For access only via Shadow Server, please enter one of the following:
 Enter VSAM DDNAME  ________
 or    VSAM DSN     ____________________________________________

 To allow access and update via the CICS Transaction Server please
 enter the following:
 Connection name . . . . . . . . . . . . .  ____  (R)
 Mirror transaction name . . . . . . . . .  ____  (R)
 Base Cluster name as assigned to CICS .  ________  (R)
 ALternate path name as assigned to CICS  ________  (O)


Press Enter to continue or  END to return to previous display.
```

Figure 15–8. Shadow OS/390 Web Server VSAM Extract

7. Enter the following information:

   a. For Shadow_VSAM, either:

   - Enter the VSAM DDName.
   - Enter the VSAM Data set name.

   The DDName can be any name you choose as long as it does not conflict with any other DDName in use by Shadow Web Server.

   Note:
   If you enter a VSAM DDName, the data set has to be assigned to DDNAME in the Shadow startup JCL statement. If you enter a VSAM DSN, it will be dynamically allocated during execution of the query.

   b. **For Shadow_VSAM for CICS**, enter:

   - **Connection name.** The CICS connection name defined in the SWS EXEC.
   - **Mirror transaction name.** The transaction id defined in CICS.
   - **Base cluster name as assigned to CICS.** The filename for the dataset name assigned to CICS.
   - **Alternate path name as assigned to CICS.** The alternate path for the dataset file name assigned to CICS.

8. Press <ENTER>.

If the extract completes with no errors, the message “Extract Successful” will appear in the upper right hand corner of the Extract panel.

For more information about the Shadow Data Mapping Facility, please refer to Chapter 13, “Data Mapping Facility,” in this manual.

Defining Multiple VSAM Logical Records Within the Same Physical File

SHADOW_VSAM and SHADOW_VSAM for CICS support multiple logical records within the same physical file by defining different views into the VSAM physical file. This is done by creating different maps containing the different views.

For example, Figure 15–9 shows two logical records within the same VSAM physical file. One view contains an individual’s demographic information, while the second contains account information. The column (field) `R ECORD_TYPE` depicts which view is present for each row (record) in the VSAM table (file).

Record 1

ACCOUNT_NUMBER	RECORD_TYPE	NAME	ADDRESS
123456789	1	DOE,JOHN	SOMEWHERE USA

Record 2

ACCOUNT_NUMBER	RECORD_TYPE	ACCOUNT_BALANCE
123456789	2	254.67

Figure 15–9. Two Logical Records within the Same VSAM Physical File

Normally a COBOL application that reads this data distinguishes the records’ content by using a record type (or view) indicator and then the redefinition of the record layout, respectively.

If the COBOL program uses a redefine of the data area, then the extracted data map will also contain the redefined columns (fields). The client application can check the content of `RECORD_TYPE` and use the appropriate columns for viewing the data.

An Alternate Approach

Place the record views into two separate data mapping definitions. Both data maps can refer to the same physical file, but each map will have different table names to distinguish their view in the VSAM dataset.

For example, create the data map, DEMOGRAF, which contains definitions for ACCOUNT_NUMBER, RECORD_TYPE, NAME, and ADDRESS. Create a second data map, ACCOUNT, which contains ACCOUNT_NUMBER, RECORD_TYPE and ACCOUNT_BALANCE. The client application can then issue the following types of queries to obtain all of the rows (records) in each view:

```
SELECT * FROM DEMOGRAF WHERE  RECORD_TYPE = 1
SELECT * FROM ACCOUNT WHERE RECORD_TYPE = 2
```

To Alternate Views

To alternate views, the application can do the following, in which the &VALUE information is substituted from the prior query ACCOUNT_NUMBER column:

```
SELECT * FROM DEMOGRAPH WHERE RECORD_TYPE = 1
SELECT * FROM ACCOUNT WHERE ACCOUNT_NUMBER =  “&VALUE” AND
RECORD_TYPE = “2”
```

Using Alternate Indexes for a VSAM Cluster

SHADOW_VSAM and SHADOW_VSAM for CICS support use of VSAM alternate indexes. This is accomplished by defining a data map that contains the following:

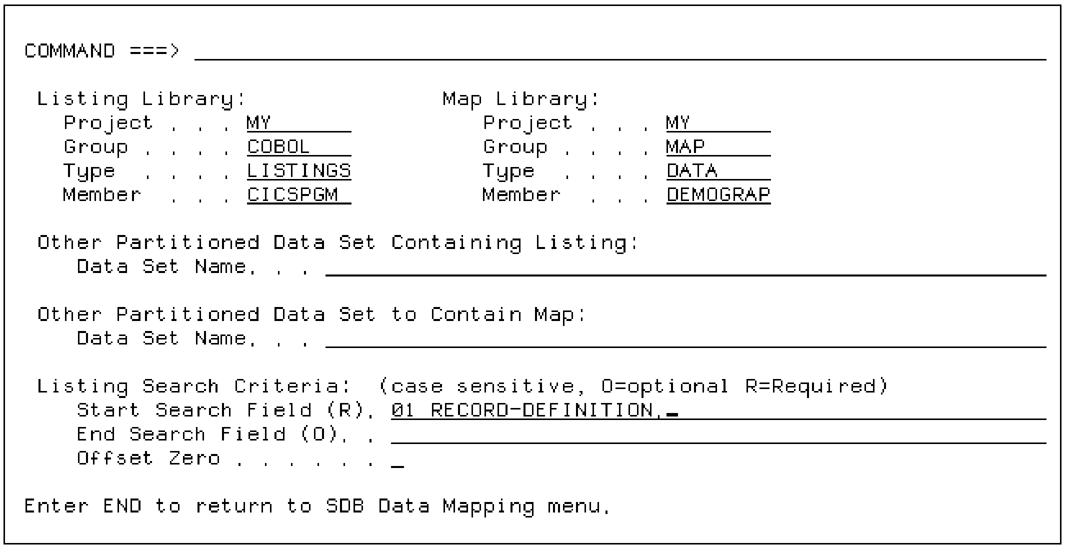
- **For SHADOW_VSAM:** This is the path name into the base VSAM cluster.
- **For SHADOW_VSAM for CICS:** This is the base cluster id and an alternate index path id as it is known to CICS.

Shadow VSAM

The Data Mapping Facility allows for the same or different views into a VSAM file by changing the map name.

Example:

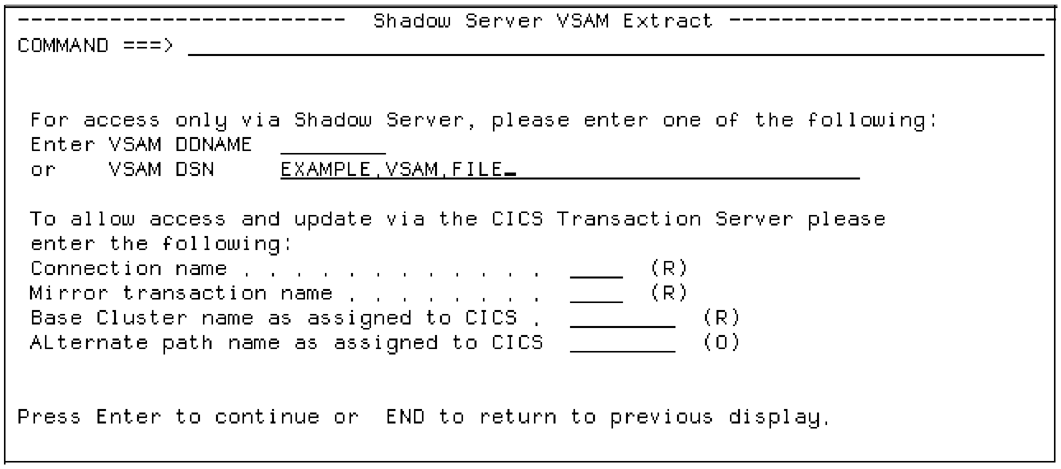
The VSAM file EXAMPLE.VSAM.FILE’s definition can be extracted from a COBOL program using the 01 RECORD-DEFINITION data definition layout imbedded in the application. This is shown in Figure 15–10.



```
COMMAND ===> ________________________________________________________________

 Listing Library:                Map Library:
   Project . . . MY______          Project . . . MY______
   Group . . . . COBOL___          Group . . . . MAP_____
   Type  . . . . LISTINGS          Type  . . . . DATA____
   Member  . . . CICSPGM_          Member  . . . DEMOGRAP

 Other Partitioned Data Set Containing Listing:
    Data Set Name. . . _______________________________________________________

 Other Partitioned Data Set to Contain Map:
    Data Set Name. . . _______________________________________________________

 Listing Search Criteria:  (case sensitive, O=optional R=Required)
    Start Search Field (R). 01 RECORD-DEFINITION._____________________________
    End Search Field (O). . __________________________________________________
    Offset Zero . . . . . . _

Enter END to return to SDB Data Mapping menu.
```

Figure 15–10. Shadow_VSAM: Record-Definition

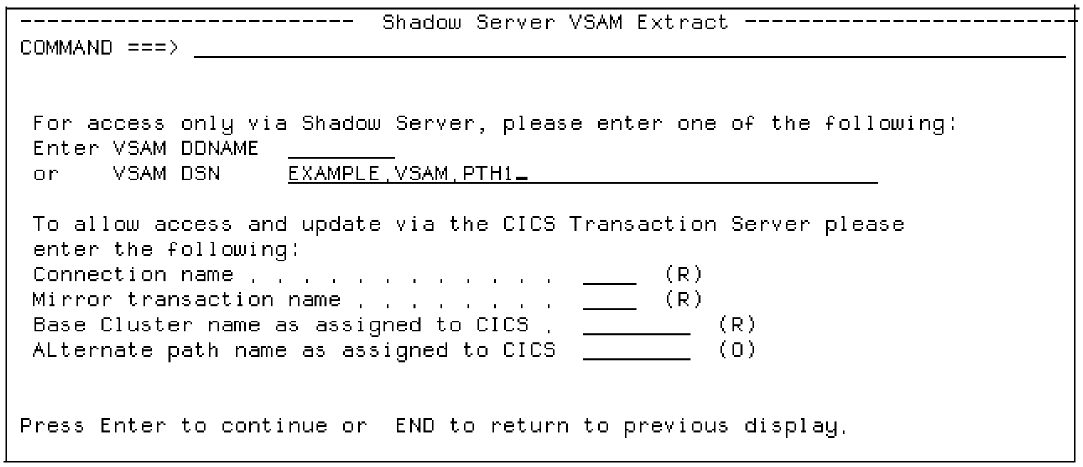
The map name assigned to this extract is EXAMVSAM. When prompted, enter the dataset name EXAMPLE.VSAM.FILE. (Figure 15–11).



```
-------------------------  Shadow Server VSAM Extract ------------------------
COMMAND ===> _________________________________________________________________


 For access only via Shadow Server, please enter one of the following:
 Enter VSAM DDNAME   ________
 or    VSAM DSN      EXAMPLE.VSAM.FILE_____________________________

 To allow access and update via the CICS Transaction Server please
 enter the following:
 Connection name . . . . . . . . . . . . .  ____  (R)
 Mirror transaction name . . . . . . . . .  ____  (R)
 Base Cluster name as assigned to CICS .  ________  (R)
 ALternate path name as assigned to CICS  ________  (O)


Press Enter to continue or  END to return to previous display.
```

Figure 15–11. Shadow_VSAM: Example.VSAM.File

To allow access using the alternate index, a second extract is performed using the same data definition layout with the map name of EXAMPATH. When prompted for the dataset name, give the alternate index path (for example, EXAMPLE.VSAM.PTH1). See Figure 15–12.



```
-------------------------  Shadow Server VSAM Extract ------------------------
COMMAND ===> ____________________________________________________________________


 For access only via Shadow Server, please enter one of the following:
 Enter VSAM DDNAME   ________
 or    VSAM DSN      EXAMPLE.VSAM.PTH1_

 To allow access and update via the CICS Transaction Server please
 enter the following:
 Connection name . . . . . . . . . . . . .  ____  (R)
 Mirror transaction name . . . . . . . . .  ____  (R)
 Base Cluster name as assigned to CICS .  ________  (R)
 ALternate path name as assigned to CICS  ________  (O)


Press Enter to continue or  END to return to previous display.
```

Figure 15–12. Shadow_VSAM: Example.VSAM.PTH1

Use the following SELECT statement to read all records and all fields on the file using the normal key:

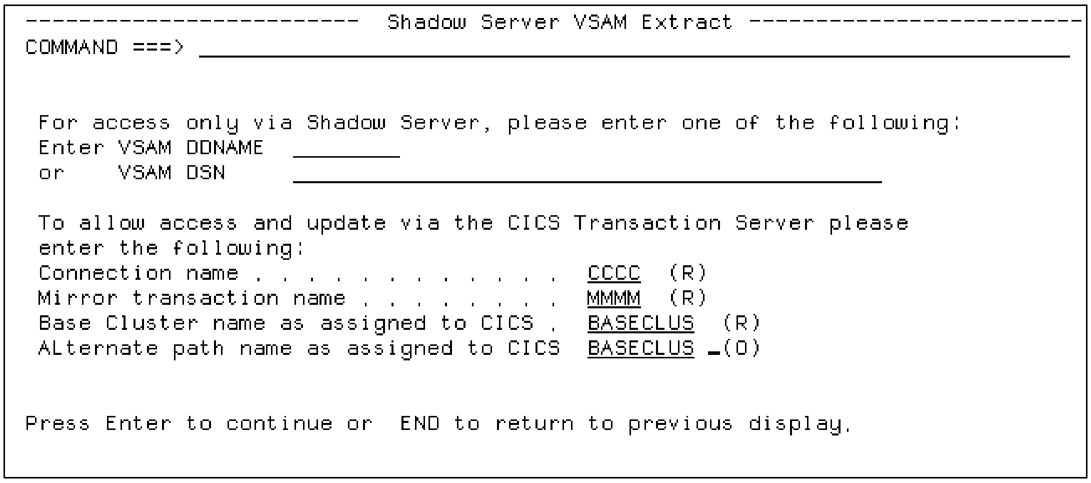
```
SELECT * FROM EXAMVSAM
```

Use the following SELECT statement to read all records and all fields on the file using the alternate index:

```
SELECT * FROM EXAMPATH
```

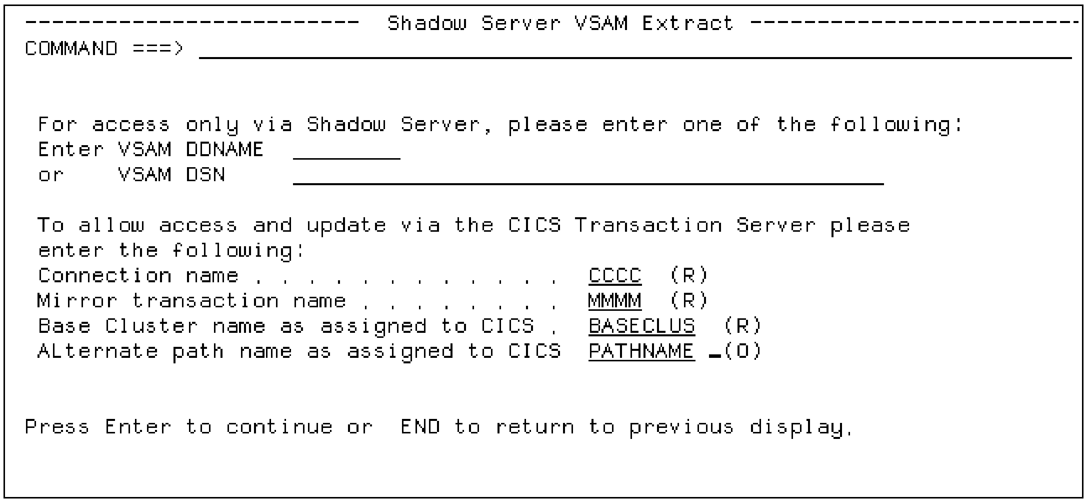
Shadow_VSAM for CICS

SHADOW_VSAM for CICS supports use of VSAM alternate indexes. This is accomplished by defining a data map that contains the base cluster id and an alternate index path id as known to CICS (see Figure 15–13 for the base cluster id, and Figure 15–14 for the alternate index path id).



```
-------------------------  Shadow Server VSAM Extract ------------------------
COMMAND ===> ____________________________________________________________________


 For access only via Shadow Server, please enter one of the following:
 Enter VSAM DDNAME   ________
 or    VSAM DSN      ____________________________________________

 To allow access and update via the CICS Transaction Server please
 enter the following:
 Connection name . . . . . . . . . . . . .  CCCC  (R)
 Mirror transaction name . . . . . . . . .  MMMM  (R)
 Base Cluster name as assigned to CICS .  BASECLUS  (R)
 ALternate path name as assigned to CICS  BASECLUS _(O)


Press Enter to continue or  END to return to previous display.
```

Figure 15–13. Shadow_VSAM for CICS: Base Cluster ID



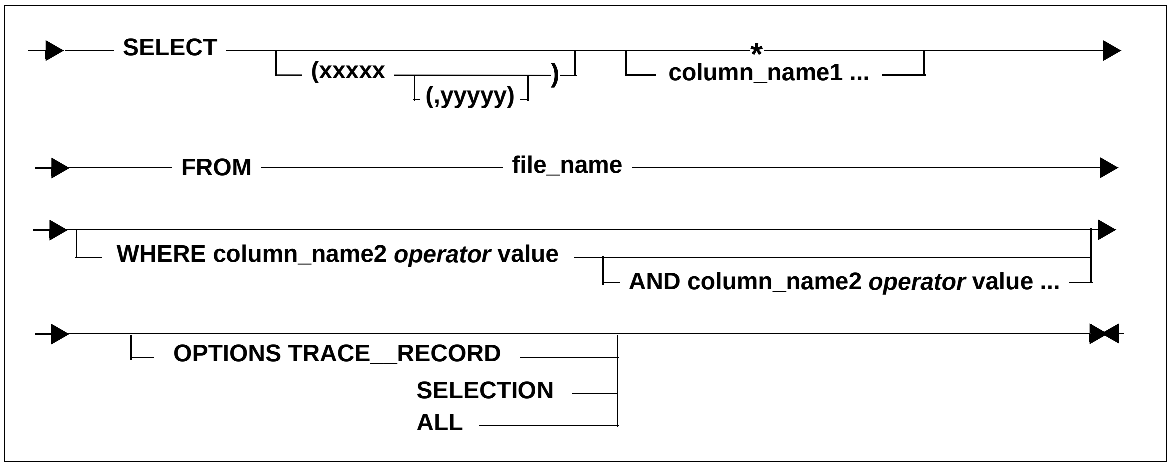
```
-------------------------  Shadow Server VSAM Extract -------------------------
COMMAND ===> ____________________________________________________________________


 For access only via Shadow Server, please enter one of the following:
 Enter VSAM DDNAME  ________
 or    VSAM DSN     ____________________________________________

 To allow access and update via the CICS Transaction Server please
 enter the following:
 Connection name . . . . . . . . . . . . .  CCCC  (R)
 Mirror transaction name . . . . . . . . .  MMMM  (R)
 Base Cluster name as assigned to CICS .  BASECLUS  (R)
 ALternate path name as assigned to CICS  PATHNAME _(O)


Press Enter to continue or  END to return to previous display.
```

Figure 15–14. Shadow_VSAM for CICS: Alternate Index Path ID

SQL Supported Syntax

Shadow_VSAM

The following syntax diagram is supported by Shadow_VSAM:



Where:

xxxxx

Is a numeric value that allows a specific number of rows to be returned.

yyyyy

Is a numeric value that allows positioning within the VSAM file starting with a specific record number in the file relative to the beginning of the file.

*****

Implies that all columns “enabled” in the Data Map will be returned.

column_name1

Can be specified if specific column(s) are to be returned.

file_name

Is the logical name associated with the VSAM file as defined in the Data Mapping Facility.

WHERE

Limits the result set to the specific records that meet the specified selection criteria.

column_name2

Is the name of the column that will be interrogated by the selection process.

(*)operator

Can have the following values:

- EQ = equal to
- NE ^= not equal
- GE >= greater than or equal
- LE<= less than or equal
- LT < less than
- GT > greater than

OPTIONS TRACE

The OPTIONS TRACE allows client tracing of the records being processed (RECORD) or selection process (SELECTION). If ALL is specified, both traces are produced. The query result set is not returned, however, a column named TRACE is returned with the resulting trace information.

Presently only the AND connector is allowed in the syntax, and there is no limit to the number of WHERE selection criteria.

Note:

When using these options, the resulting trace can become quite large.

Shadow_VSAM for CICS

The SQL syntax supported by Shadow_VSAM for CICS must be strictly coded as described in this section. The following syntax diagrams are supported by Shadow_VSAM for CICS:

Syntax

```
CALL SHADOW_CICS('EXVS','sql statement','optional connection name
override','optional transaction id override')
```

where sql_statement may be one of the following:

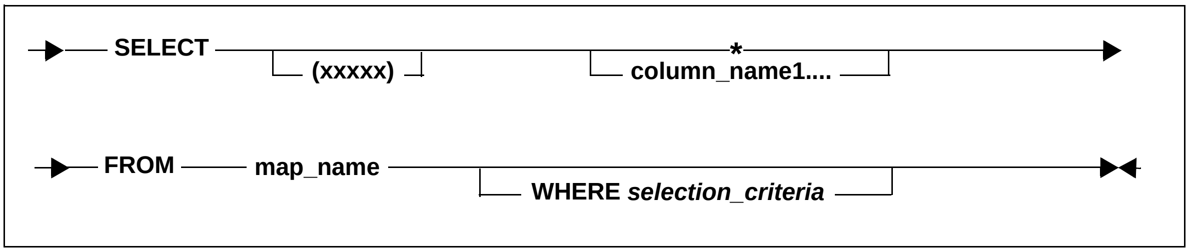
Note:

The parameters `'optional connection name override'` and `'optional transaction id override'` allow for the information contained in the map_name to be overridden for the default connection name and transaction ID. See the Shadow Programming Guide for more information about the `'CCCC'` (CICS connection name) and `'TTTT'` (CICS transaction ID) parameters.

If you use one optional parameter in the Call statement, then you must use all other optional parameters in the Call statement as well.

SELECT

The SELECT statement provides reading access of VSAM data. The syntax is:



Where:

map_name

Must be a VSAM defined datamap that contains the associated CICS FILEID assignment as well as the CICS transaction and connection names.

xxxxx

Is an optional numeric value that limits the number of selected rows returned to the indicated value.

The following examples show how to use the SELECT syntax:

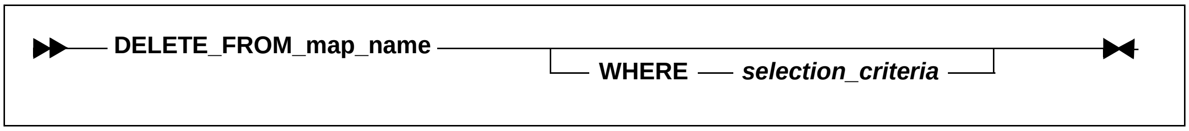
- `SELECT * FROM EMPLOYEES`
  This reads all columns in all rows (records) in the VSAM table(file) associated with the `EMPLOYEES` data map.

- `SELECT  LAST_NAME FIRST_NAME FROM EMPLOYEES WHERE LAST_NAME = "SMITH"`

  This reads rows from the VSAM file described by the `EMPLOYEES` data map and returns columns `LAST_NAME` and `FIRST_NAME` for the rows whose `LAST_NAME` column contain the value "`SMITH`".

DELETE

The DELETE statement allows for removal of rows from the VSAM file described by the map_name. The syntax is:



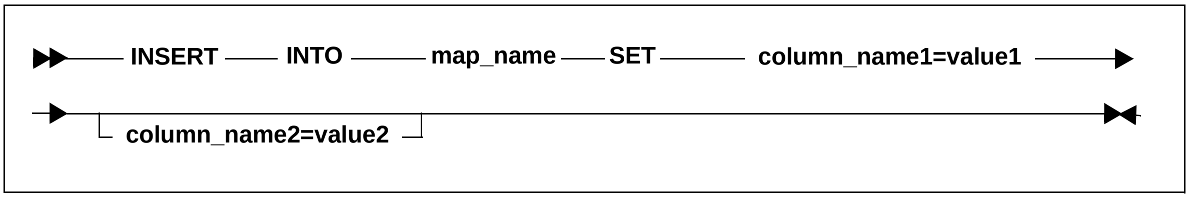
The following examples show how to use the DELETE syntax:

- `DELETE FROM EMPLOYEES`
  This deletes all rows from the VSAM file described by the data map `EMPLOYEES`

- `DELETE FROM EMPLOYEES WHERE LAST_NAME = "SMITH"`
  This deletes rows from the VSAM file described by data map `EMPLOYEES` that contain the value `SMITH` in the `LAST_NAME` column.

INSERT

The INSERT statement allows for the insertion or addition of a row into a VSAM file described by map_name. The syntax is:



The SET clause describes columns and their corresponding data values to be placed into the inserted row. This statement requires that one of the columns to be inserted describes the full VSAM primary key. When using an alternate index data map, the full primary key must still be contained in the statement as one of the columns to be inserted.

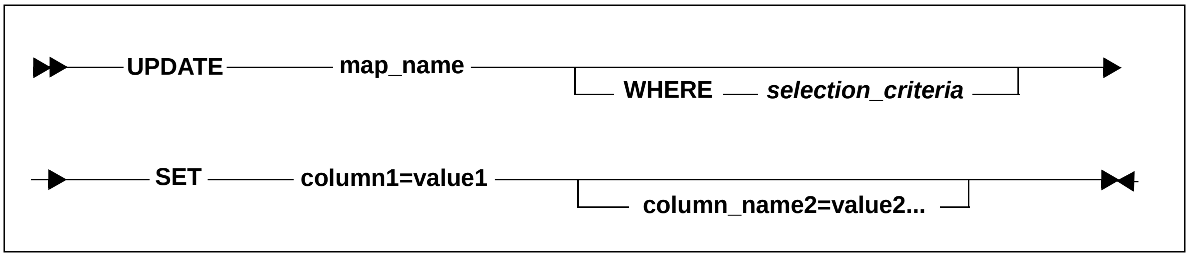
The following example shows how to use the INSERT syntax:

```
INSERT INTO EMPLOYEES SET LAST_NAME = "SMITH" FIRST_NAME = "JOHN"
```

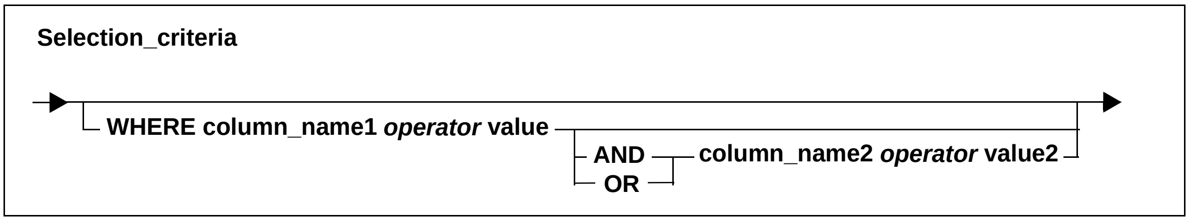
adds a new row to the VSAM file described by the data map EMPLOYEES and set the LAST_NAME to the value "SMITH" and FIRST_NAME to the value "JOHN". The column LAST_NAME is the primary key to the VSAM cluster.

UPDATE

The UPDATE statement allows for the alteration, or update, of an existing row contained in the VSAM file described by map_name. The syntax is:



Selection Criteria


Selection_criteria
WHERE column_name1 operator value
AND
OR
column_name2 operator value2

Note:
ANDs are evaluated before ORs. For example, the statement

```
column1=value1 and column2=value2 or
column3=value3 and column4=value4
```

is evaluated as:

```
(column1=value1 and column2=value2) or
(column3=value3 and column4=value4)
```

The syntax does not support parenthetical grouping of selection criteria, and is evaluated in the order that it is written.

operator

Operators can have the following values:

- EQ = equal to

- NE ^= not equal
- GE >= greater than or equal
- LE<= less than or equal
- LT < less than
- GT > greater than

Codes

The following return codes apply to Shadow_VSAM:

Return Code	Explanation
0001	Error occurred during Shadow interface. Review trace browse for determination of problem.
0003	WHERE column name is not found in data map.
1005-1006	WHERE operator is invalid.
1007	WHERE criteria is incomplete.
1008	WHERE connector value is invalid.
2000-2999	Statement syntax error. The last three digits indicate the statement keyword in question.
4001	FROM clause is not found. Check FROM criteria specification.
4002	SELECTion criterion is missing.
4003	No data maps area present in system.
4004	No enabled data map is present for file requested.
4005	A SELECT field specification has been disabled in the data map.
4020	Invalid SELECT (nnn) criteria.
5000	ACB generation error. The last three bytes of the error number is the actual system response code for the request.
6000	RPL generation error. The last three bytes of the error number is the actual system response code for the request.
7000	VSAM open error. The last three bytes of the error number is the actual system response code for the request.
8000	VSAM read error. The last three bytes of the error number is the actual system response code for the request.

The following abend codes apply to Shadow_VSAM for CICS:

Abend Code	Explanation
SD01	The Shadow VSAM access program was invoked by some means other than a Distributed Program Link (DPL) request. This could be a user error if another transaction is trying to link to the program, or a transid is defined with SDCIVSAM as its initial program.
SDO2|SD03|SD04|SDO5	The commarea passed to SDCIVSAM is in some way invalid. This can be caused by using different releases of load modules in the Shadow Server, or by a program erroneously linking to SDCIVSAM.

CHAPTER 16: Shadow Web Interface

This chapter covers the Shadow Web Interface component for Shadow OS/390 Web Server. The following Web browsers are supported:

- Netscape Navigator™ v 4.0 or higher
- Internet Explorer™ v 4.0 or higher

The Shadow Web Interface

The Shadow Web Interface is a GUI (Graphical User Interface) that allows you to perform many of the same functions available on the ISPF panels. The Shadow Web Interface:

- Enables administrative functions over the Web, which include controlling and monitoring for:
  - Product
  - Storage
  - Databases
  - CICS
  - IMS
  - RRS
  - TSO
- Is controlled using a product parameter.
- Has trace browse support.
- Integrates security features.

Before you Begin

Before you begin, you will need to:

1. Install Shadow OS/390 Web Server and the various components for your site. (See the *Shadow Installation Guide* for more information.)

2. Set the HOSTDOMAIN product parameter to allow cookies to be recognized. There must be a minimum three node name separated by periods. For example: `'P390.NEONSYS.COM'`.

3. Modify the SWIURLNAME parameter in your SWSxIN00 initialization EXEC to enable it. The default parameter value is SWICNTL. For example:

```
MODIFY PARM NAME (SWIURLNAME)      VALUE (SWICNTL)
```

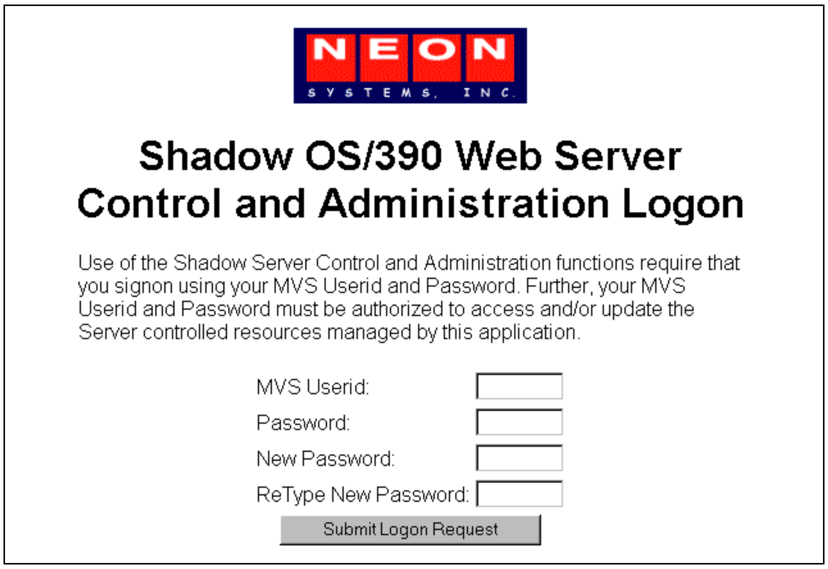
> **Note:**
> The SWIURLNAME (Shadow Web Interface URL Name) activates the Web interface. The default value, SWICNTL, is loaded during installation.

4. Verify that your browser accepts cookies. If it does not, turn on that feature.
5. Specify the URL (and port number) in your browser. For example:

   ```
   http://domain_name:portnumber/swicntl
   ```

   where swicntl is the SWIURLNAME parameter.
6. Press <ENTER>.

   The login panel appears.



Shadow OS/390 Web Server
Control and Administration Logon

Use of the Shadow Server Control and Administration functions require that you signon using your MVS Userid and Password. Further, your MVS Userid and Password must be authorized to access and/or update the Server controlled resources managed by this application.

MVS Userid:
Password:
New Password:
ReType New Password:

Figure 16–1. Shadow Web Logon Screen

Logging On

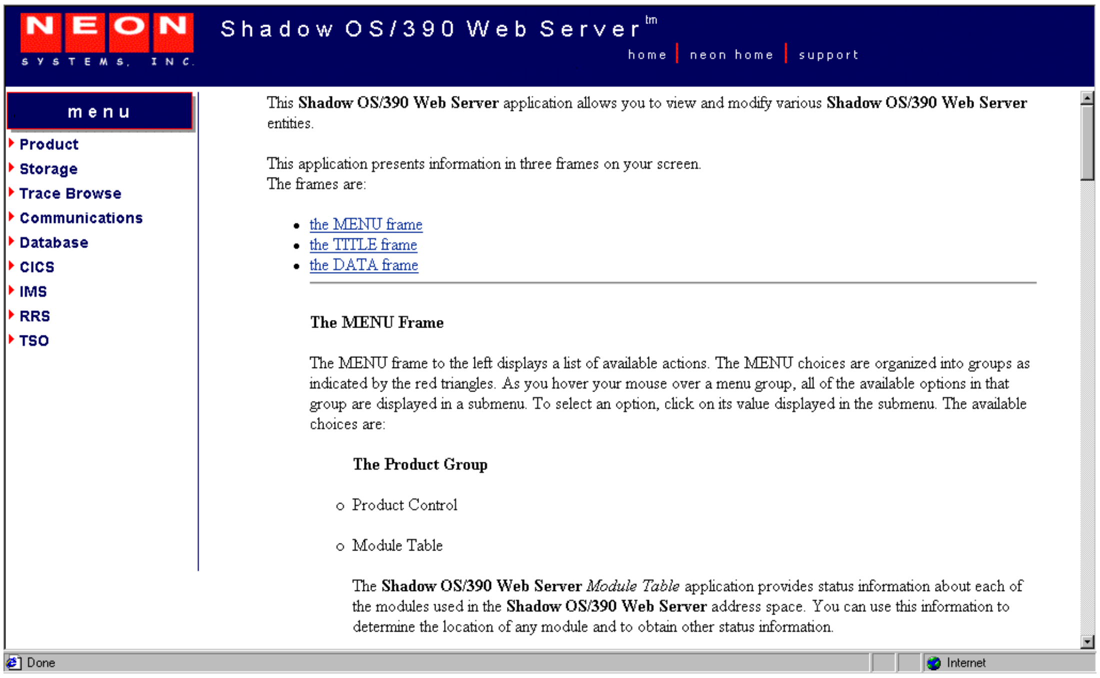
Before you can use the interface, you must first logon.

1. Enter your MVS Userid and Password.

> **Note:**
> You must wait for the entire page to display before entering any information. If you do not, security will not be properly implemented and your Userid and Password will *not* be sent.

2. Select <Submit Logon Request> or press <ENTER>.

   The Shadow Web Interface home page displays.



This **Shadow OS/390 Web Server** application allows you to view and modify various **Shadow OS/390 Web Server** entities.

This application presents information in three frames on your screen.
The frames are:

- the MENU frame
- the TITLE frame
- the DATA frame

The MENU Frame

The MENU frame to the left displays a list of available actions. The MENU choices are organized into groups as indicated by the red triangles. As you hover your mouse over a menu group, all of the available options in that group are displayed in a submenu. To select an option, click on its value displayed in the submenu. The available choices are:

The Product Group

- Product Control
- Module Table

The **Shadow OS/390 Web Server** *Module Table* application provides status information about each of the modules used in the **Shadow OS/390 Web Server** address space. You can use this information to determine the location of any module and to obtain other status information.

Figure 16–2. Shadow Web Interface Home Page

The Home Page

Information is presented in three frames on your screen.

- The TITLE frame
- The DATA frame
- The MENU frame

The TITLE Frame

The TITLE frame is at the top of your screen. It displays:

- The product title.
- A link back to the Shadow Web Interface home page (home).
- A link to NEON Systems, Inc. home page (neon home).
- A link to NEON Systems, Inc. Support Dept. (support).

The DATA Frame

The DATA frame appears in the lower right and displays the application information.

The MENU Frame

The MENU frame, which is on the left of your screen, displays a list of available actions. The MENU is organized into groups with submenus. Most of the panels have an Action column with links. These links can be used to gain further information about the row or selected item.

Product

The following submenu items are available:

Submenu	Description
Product Control	Displays a composite of statistical and general information about the product, such as subsystem names, status and addresses.
Module Table	Provides status information about each of the modules used in the Shadow OS/390 Web Server address space. This information can be used to determine the location of any module and other status information.
Parameter Groups	Allows you to control the started task parameters created using the SWSxIN00 initialization EXEC. Some of these parameters can be modified after setup. For information on viewing and changing parameters, see “Examples” on page 16-9.
Tasks	Displays current and cumulative information for monitoring and controlling specific Shadow OS/390 Web Server Tasks. With it, you can kill a selected task, display the task trace control block or user detail for the selected row.
Process Block	Displays information on process blocks, such as name, origin, usage and addresses.
Token Control	Allows you to display and control Shadow OS/390 Web Server execution tokens. Using this application you can determine the status of a token, look at token data, and kill tokens, as needed.
MIME/Filetype Table	Provides status information about each of the entries in the configurable Shadow OS/390 Web Server MIME table. You can use this information to determine if a filetype entry has been defined, obtain status information, and update entries.
Data Mapping Block	Allows you to view the Mapping Data Structures.
Dbcs Translate Table	Displays the DBCS Translation tables in either an ASCII-to-EBCDIC DBCS translation table or EBCDIC-to-ASCII DBCS translation table.

Storage

The storage option is a virtual storage information application that allows you to see the allocation of private virtual storage in Shadow OS/390 Web Server’s address space. This includes:

- Who is using the Private and Extended areas
- Which programs are being run in it.

The storage option is designed to help you locate potential problems areas. The following submenu items are available:

Submenu	Description
Internal Control Blocks	Displays internal product control blocks and storage areas as formatted lists with descriptions and as a hexadecimal dump. You must have MVS Security Subsystem read authorization to the Servers's 'CONTROLBLOCKS' generalized resource rule in order to use this diagnostic function.
Pvt Area Stg Display	Displays the allocation of virtual storage information in the server's address space by subpool. This includes the amount of storage: • Allocated to a subpool. • Used (Allocated - Free). • Not used (Free).
Common Area Stg Display	Displays summary information of the allocation of virtual storage for each subpool in the server's address space. This includes the amount of storage: • Allocated to a subpool. • Used (Allocated - Free). • Not used (Free).
TCB Storage Summary	Displays summary information pertaining to the allocation of virtual storage for each TCB in the server's address space. This includes the TCB address and the amount of owned storage: • Allocated. • Used (Allocated - Free). • Not used (Free).
Allocated Storage	Displays an overview of the allocation of virtual storage in the server's address space. The information is displayed by regions in which the address and size of the region is reported. Within each region, the address and size of each block of allocated virtual storage is displayed.
Virtual Storage	Displays an overview of both allocated and unallocated virtual storage in the server's address space. This information is displayed by regions in which the address and size of the region is reported. Within each region, the address and size of each block of allocated and unallocated virtual storage is displayed.
Unallocated Storage	Displays an overview of unallocated virtual storage in the server's address space. This information is displayed by regions in which the address and size of the region is reported. Within each region, the address and size of each block of unallocated virtual storage is displayed.

Trace Browse

The trace list, which is maintained by the SWS started task, is a record of all communication, WWW, and internal events in message format. The most recent messages are at the bottom of the list and the oldest messages are at the top. The list is implemented as a FIFO buffer, the size of which is an SWS parm. When the list is full, messages are removed from the top of the buffer to make room for the newer messages at the bottom.

The following submenu items are available:

Submenu	Description
Trace Browse Records	Displays trace browse records. When you select the Trace Browse Records menu option, the panel displays the bottom of the trace list. This contains the most recent additions. See Figure 16–11 and Figure 16–12.
Trace Browse Control	Allows you to control what trace browse records. With it you can limit the display of record types as well as set the number of records to be retrieved during each interaction. See Figure 16–13 through Figure 16–16 for examples of the Trace Browse Control screens.

Communications

The Communications menu allows you to:

- Display and control the link table.
- Display and modify in-flight transactions (remote users).

The following submenu items are available:

Submenu	Description
Link Control	Allows you to displays and control teleprocessing links. Use this application to determine and change the status of the links.
IP Address Tree	Displays the Internet Protocol network address of a node.
Remote Users	Displays current and cumulative transactions regarding users on remote nodes. Remote users connect with the local Shadow OS/390 Web Server to access databases on the local node.

Database

The database submenu is used to view and modify the Web Server's database table. The following submenu item is available:

Submenu	Description
Database Control	Allows you to view and modify the Shadow OS/390 Web Server database table. With it, you can view database statistics, such as version number, if the database is up or down, and address. Plus, you can clear pending requests.

CICS

This is the CICS Control Facility. The following submenu items are available:

Submenu	Description
CICS Connections	Allows you to monitor and control CICS connections. The main CICS Connections panel is summarized by connection name. It contains information such as access method, total sessions, and protocol used. You can use the <Sessions> link to drill down to view individual sessions or to change the status (ANY, UP, or DOWN) or you can change the status on the main CICS Connections panel and apply it to all the sessions for the connection name.
CICS Session	Allows you to monitor and control each CICS sessions.

IMS

The IMS Control Facility allows you to monitor and control your access to IMS/TM and IMS/DB. An APPC/MVS provides the ability to monitor APPC/MVS conversations to IMS, in real-time and historical mode. APPC/MVS conversations can be terminated automatically (inactivity timeout setting), or by manual intervention (line command).

The following submenu items are available:

Submenu	Description
IMS LTERM Table	Allows you to display and control LTERM mapping. With it, you can exploit existing IMS LTERM security by assigning known LTERM names to inbound IMS transactions based upon either the userid or the IP address of the originating requestor.
APPC/MVS Detail	Allows you to monitor APPC/MVS conversations to IMS in historical mode, such as userid, conversation start time, total sends, and total data received.
APPC/MVS Realtime	Allows you to monitor APPC/MVS conversations to IMS in real-time mode.
APPC/MVS Interval	Displays a summary of APPC/MVS Conversation Statistics.

TSO

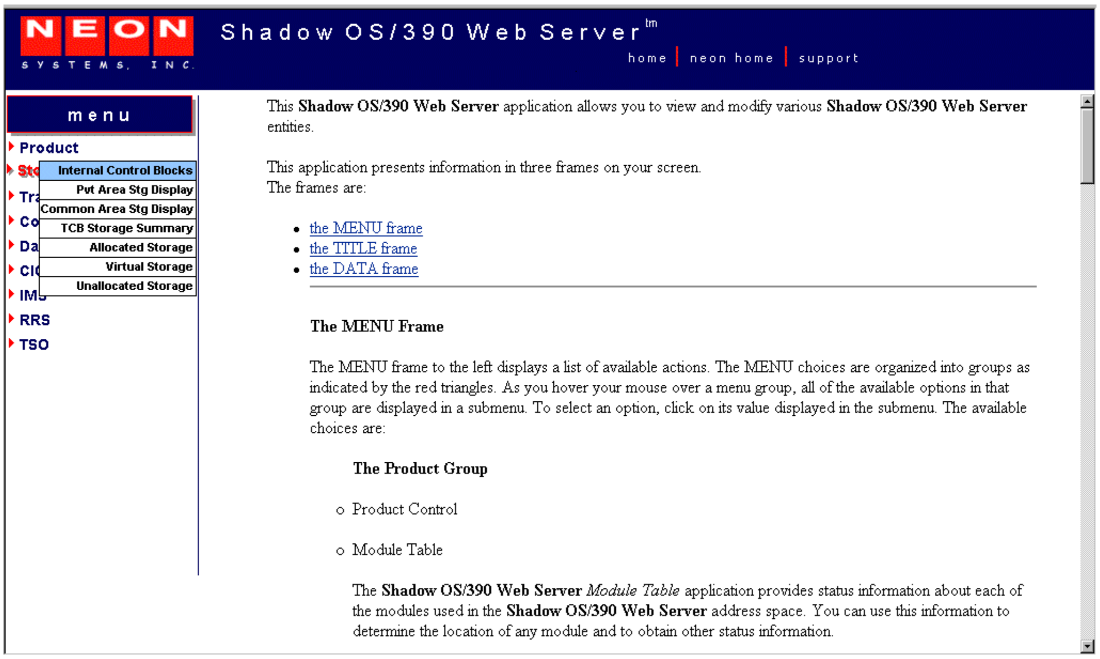
The following submenu item is available:

Submenu	Description
TSO Servers	Displays TSO server status, such as server status, job names, and address space.

Using the MENUs

To use the menu:

1. Move your mouse over a menu group to display all the available option in that submenu. For example, the following shows the submenu for storage:



This **Shadow OS/390 Web Server** application allows you to view and modify various **Shadow OS/390 Web Server** entities.

This application presents information in three frames on your screen.
The frames are:

- the MENU frame
- the TITLE frame
- the DATA frame

The MENU Frame

The MENU frame to the left displays a list of available actions. The MENU choices are organized into groups as indicated by the red triangles. As you hover your mouse over a menu group, all of the available options in that group are displayed in a submenu. To select an option, click on its value displayed in the submenu. The available choices are:

The Product Group

- Product Control
- Module Table

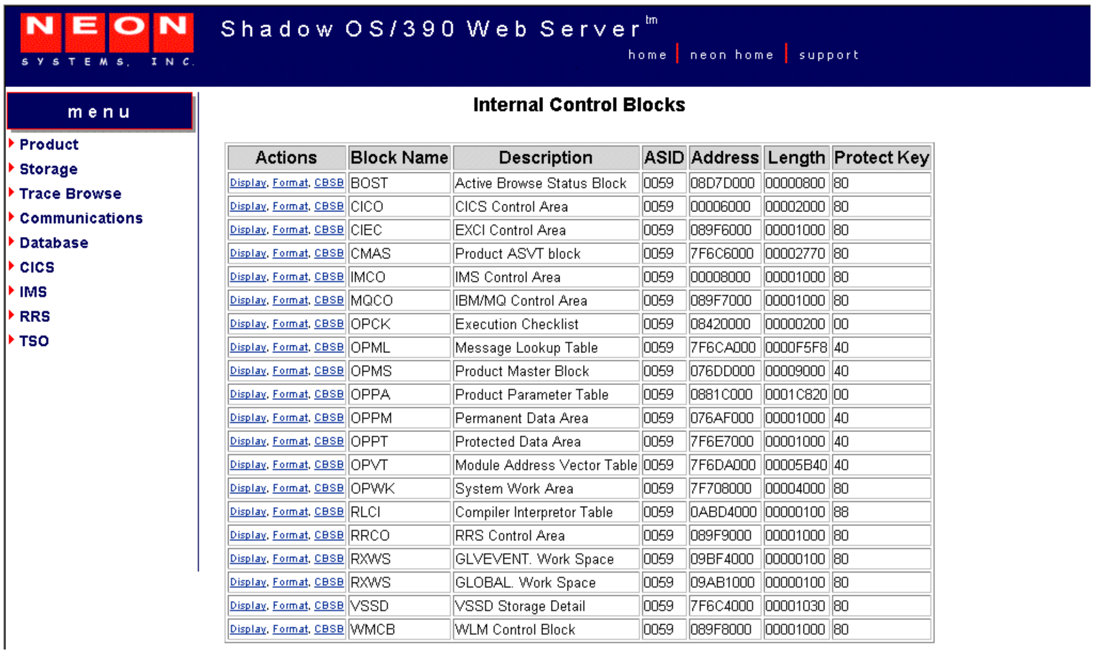
The **Shadow OS/390 Web Server** *Module Table* application provides status information about each of the modules used in the **Shadow OS/390 Web Server** address space. You can use this information to determine the location of any module and to obtain other status information.

Figure 16–3. An Example where Submenus are Displayed

Note:
When the mouse is over a menu item (or a link), a brief description appears on the lower left corner of the status bar. In Figure 16–3 a definition is displayed for Internal Control Block.

2. Click to select the highlighted value displayed in the submenu.

3. Edit/view the information in the DATA frame.



Internal Control Blocks

Actions	Block Name	Description	ASID	Address	Length	Protect Key
Display, Format, CBSB	BOST	Active Browse Status Block	0059	08D7D000	00000800	80
Display, Format, CBSB	CICO	CICS Control Area	0059	00006000	00002000	80
Display, Format, CBSB	CIEC	EXCI Control Area	0059	089F6000	00001000	80
Display, Format, CBSB	CMAS	Product ASVT block	0059	7F6C6000	00002770	80
Display, Format, CBSB	IMCO	IMS Control Area	0059	00008000	00001000	80
Display, Format, CBSB	MQCO	IBM/MQ Control Area	0059	089F7000	00001000	80
Display, Format, CBSB	OPCK	Execution Checklist	0059	08420000	00000200	00
Display, Format, CBSB	OPML	Message Lookup Table	0059	7F6CA000	0000F5F8	40
Display, Format, CBSB	OPMS	Product Master Block	0059	076DD000	00009000	40
Display, Format, CBSB	OPPA	Product Parameter Table	0059	0881C000	0001C820	00
Display, Format, CBSB	OPPM	Permanent Data Area	0059	076AF000	00001000	40
Display, Format, CBSB	OPPT	Protected Data Area	0059	7F6E7000	00001000	40
Display, Format, CBSB	OPVT	Module Address Vector Table	0059	7F6DA000	00005B40	40
Display, Format, CBSB	OPWK	System Work Area	0059	7F708000	00004000	80
Display, Format, CBSB	RLCI	Compiler Interpretor Table	0059	0ABD4000	00000100	88
Display, Format, CBSB	RRCO	RRS Control Area	0059	089F9000	00001000	80
Display, Format, CBSB	RXWS	GLVEVENT. Work Space	0059	09BF4000	00000100	80
Display, Format, CBSB	RXWS	GLOBAL. Work Space	0059	09AB1000	00000100	80
Display, Format, CBSB	VSSD	VSSD Storage Detail	0059	7F6C4000	00001030	80
Display, Format, CBSB	WMCB	WLM Control Block	0059	089F8000	00001000	80

Figure 16–4. A Submenu Item on the Storage Menu

4. Use the links in the Actions column to obtain more information on the internal control blocks.

   – Display - Displays the selected product control block or storage area.
   – Format - Formats the selection list entry vertically.
   – CBSB - Displays the CBSB control block for the selected row.

Note:
When the mouse is over a link (or menu item), a brief description appears on the lower left corner of the status bar. In Figure 16–4 the mouse is over the link CBSB.

Examples

The following examples show you how to:

- Drill down through the panels.
- Update (change) a panel.

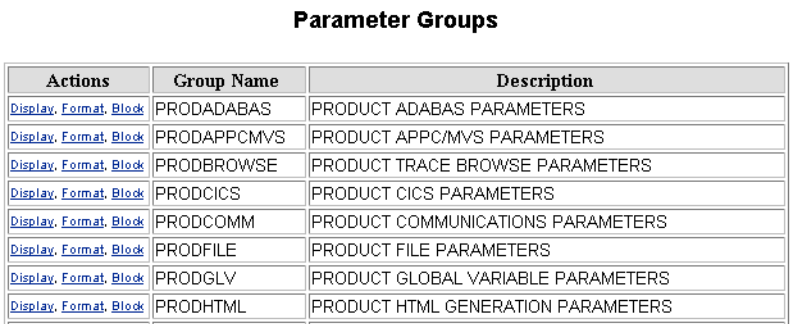
Drilling Down through the Panels

This example starts with the main panel, Product, then drills down through the data to obtain an explanation of the parameter.

```
Product > Parameter Groups > Display > MSG.
```

1. Select the menu item <Product>.
2. Select <Parameter Groups>.

   The Parameter Groups panel is displayed.

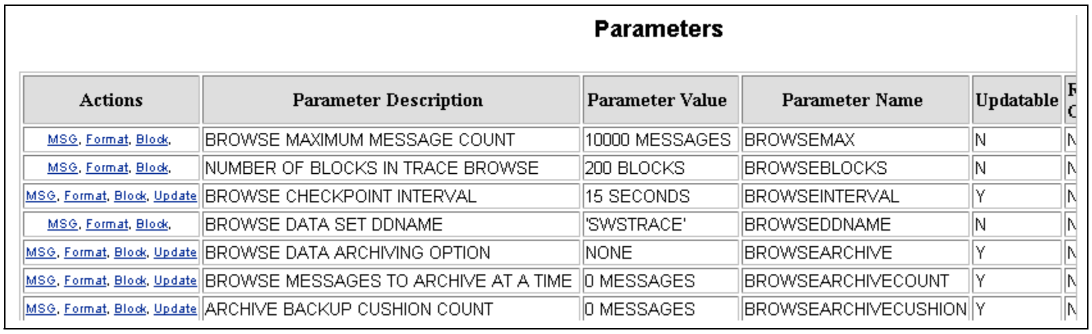


Parameter Groups

Actions	Group Name	Description
Display, Format, Block	PRODADABAS	PRODUCT ADABAS PARAMETERS
Display, Format, Block	PRODAPPCMVS	PRODUCT APPC/MVS PARAMETERS
Display, Format, Block	PRODBROWSE	PRODUCT TRACE BROWSE PARAMETERS
Display, Format, Block	PRODCICS	PRODUCT CICS PARAMETERS
Display, Format, Block	PRODCOMM	PRODUCT COMMUNICATIONS PARAMETERS
Display, Format, Block	PRODFILE	PRODUCT FILE PARAMETERS
Display, Format, Block	PRODGLV	PRODUCT GLOBAL VARIABLE PARAMETERS
Display, Format, Block	PRODHTML	PRODUCT HTML GENERATION PARAMETERS

Figure 16–5. Parameter Groups Panel

3. Click the <Display> link in the Actions column to see individual members in the group.

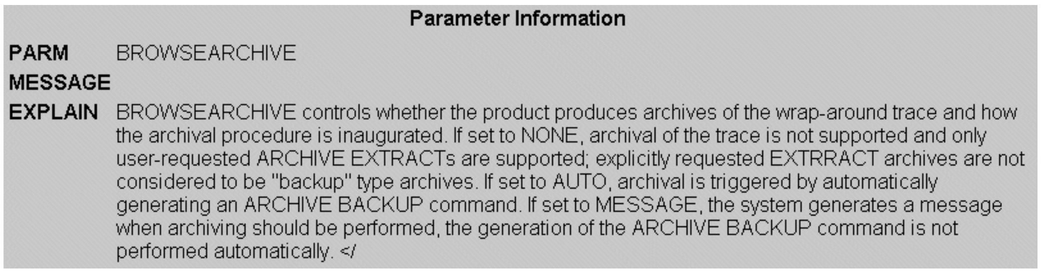


Parameters

Actions	Parameter Description	Parameter Value	Parameter Name	Updatable	F C
MSG, Format, Block,	BROWSE MAXIMUM MESSAGE COUNT	10000 MESSAGES	BROWSEMAX	N	N
MSG, Format, Block,	NUMBER OF BLOCKS IN TRACE BROWSE	200 BLOCKS	BROWSEBLOCKS	N	N
MSG, Format, Block, Update	BROWSE CHECKPOINT INTERVAL	15 SECONDS	BROWSEINTERVAL	Y	N
MSG, Format, Block,	BROWSE DATA SET DDNAME	'SWSTRACE'	BROWSEDDNAME	N	N
MSG, Format, Block, Update	BROWSE DATA ARCHIVING OPTION	NONE	BROWSEARCHIVE	Y	N
MSG, Format, Block, Update	BROWSE MESSAGES TO ARCHIVE AT A TIME	0 MESSAGES	BROWSEARCHIVECOUNT	Y	N
MSG, Format, Block, Update	ARCHIVE BACKUP CUSHION COUNT	0 MESSAGES	BROWSEARCHIVECUSHION	Y	N

Figure 16–6. Trace Browse Parameter List

4. Select <MSG> from the Actions column. In this example, BROWSEARCHIVE, an updatable parameter was selected.

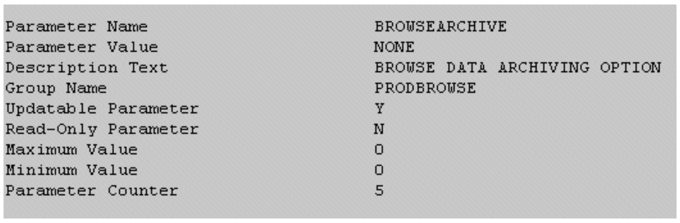


Parameter Information

PARM	BROWSEARCHIVE
MESSAGE	
EXPLAIN	BROWSEARCHIVE controls whether the product produces archives of the wrap-around trace and how the archival procedure is inaugurated. If set to NONE, archival of the trace is not supported and only user-requested ARCHIVE EXTRACTs are supported; explicitly requested EXTRRACT archives are not considered to be "backup" type archives. If set to AUTO, archival is triggered by automatically generating an ARCHIVE BACKUP command. If set to MESSAGE, the system generates a message when archiving should be performed, the generation of the ARCHIVE BACKUP command is not performed automatically. </

Figure 16–7. A Sample MSG Screen

This panel, and any other panel that is totally gray, cannot be updated. They are for information purposes only. For example, Format displays the information for the selected row.



```
Parameter Name                    BROWSEARCHIVE
Parameter Value                   NONE
Description Text                  BROWSE DATA ARCHIVING OPTION
Group Name                        PRODBROWSE
Updatable Parameter               Y
Read-Only Parameter               N
Maximum Value                     0
Minimum Value                     0
Parameter Counter                 5
```

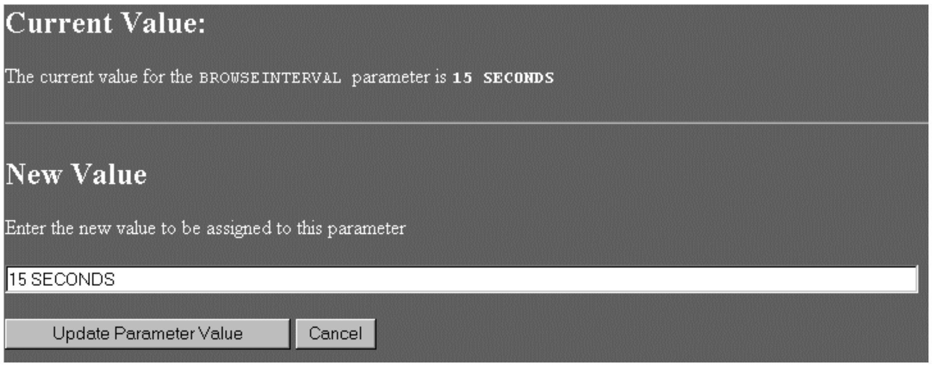
Figure 16–8. A Sample Format Screen

Updating a Parameter

To determine if a parameter can be updated, either look for an Update link in the Actions column or check the “Updatable” column. (See Figure 16–6.) If the parameter is not updatable (N), then an Update link will not appear in the Actions column.

1. Select <Update> link from the Actions column. (See Figure 16–6.)

   The following screen appears:



Current Value:

The current value for the BROWSEINTERVAL parameter is **15 SECONDS**

New Value

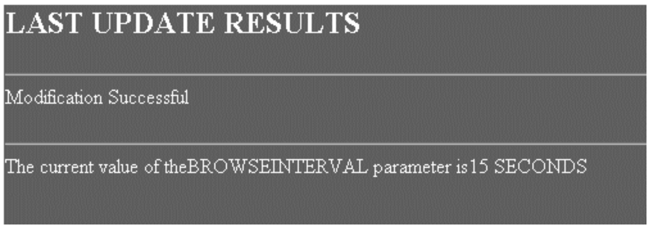
Enter the new value to be assigned to this parameter

15 SECONDS

Figure 16–9. Update Parameter Panel

2. Enter the value.

3. Select <Update Parameter Value> or <Cancel>.


LAST UPDATE RESULTS

Modification Successful

The current value of theBROWSEINTERVAL parameter is15 SECONDS

Figure 16–10. Updated Parameter Screen

Additional Screens

There are a few additional screens that do not match the above types.

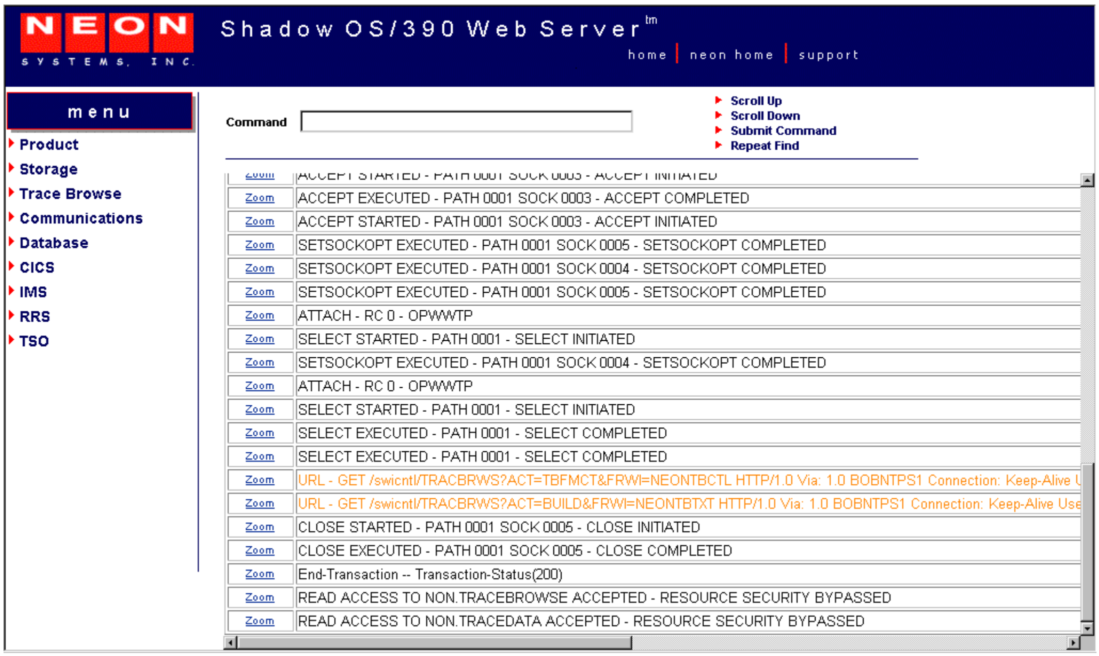
Trace Browse Records

The submenu, Trace Browse Records, allows you to view the Trace Browse records beginning at the *bottom* of the panel. The scroll bar on the far right will be positioned at the bottom. This is the default position when you access this panel.

This panel also contains additional scroll and command features.

When you use these commands, the display will be placed at the top of the browser frame and not at the bottom as it does initially.

- **Command.** Use this to enter your Trace Browse command, then select <Submit the Command> or press <ENTER>. The valid Trace Browse commands are DISPLAY, LOCATE, FIND and RFIND. They are discussed in the next section.
- **Scroll Up.** Scrolls the Trace Browse display up.
- **Scroll Down.** Scrolls the Trace Browse display down.
- **Repeat Find.** Repeats the most recent **FIND** command.



```
Zoom  ACCEPT STARTED - PATH 0001 SOCK 0003 - ACCEPT INITIATED
Zoom  ACCEPT EXECUTED - PATH 0001 SOCK 0003 - ACCEPT COMPLETED
Zoom  ACCEPT STARTED - PATH 0001 SOCK 0003 - ACCEPT INITIATED
Zoom  SETSOCKOPT EXECUTED - PATH 0001 SOCK 0005 - SETSOCKOPT COMPLETED
Zoom  SETSOCKOPT EXECUTED - PATH 0001 SOCK 0004 - SETSOCKOPT COMPLETED
Zoom  SETSOCKOPT EXECUTED - PATH 0001 SOCK 0005 - SETSOCKOPT COMPLETED
Zoom  ATTACH - RC 0 - OPWWTP
Zoom  SELECT STARTED - PATH 0001 - SELECT INITIATED
Zoom  SETSOCKOPT EXECUTED - PATH 0001 SOCK 0004 - SETSOCKOPT COMPLETED
Zoom  ATTACH - RC 0 - OPWWTP
Zoom  SELECT STARTED - PATH 0001 - SELECT INITIATED
Zoom  SELECT EXECUTED - PATH 0001 - SELECT COMPLETED
Zoom  SELECT EXECUTED - PATH 0001 - SELECT COMPLETED
Zoom  URL - GET /swicntl/TRACBRWS?ACT=TBFMCT&FRWI=NEONTBCTL HTTP/1.0 Via: 1.0 BOBNTPS1 Connection: Keep-Alive U
Zoom  URL - GET /swicntl/TRACBRWS?ACT=BUILD&FRWI=NEONTBTXT HTTP/1.0 Via: 1.0 BOBNTPS1 Connection: Keep-Alive Use
Zoom  CLOSE STARTED - PATH 0001 SOCK 0005 - CLOSE INITIATED
Zoom  CLOSE EXECUTED - PATH 0001 SOCK 0005 - CLOSE COMPLETED
Zoom  End-Transaction -- Transaction-Status(200)
Zoom  READ ACCESS TO NON.TRACEBROWSE ACCEPTED - RESOURCE SECURITY BYPASSED
Zoom  READ ACCESS TO NON.TRACEDATA ACCEPTED - RESOURCE SECURITY BYPASSED
```

Figure 16–11. An Example of the Submenu Trace Browse Records

The Trace Browse Commands

The following are valid Trace Browse commands:

Command	Abbreviation	Function

DISPLAY	D	Used to control the formatted columns of the display. Up to 5 columns can be displayed at one time.
		Syntax example:
		`D column-name column-name ...`
		Example:
		`DISPLAY DATE TIME USERID ASID SECONDS`

Command	Abbreviation	Function

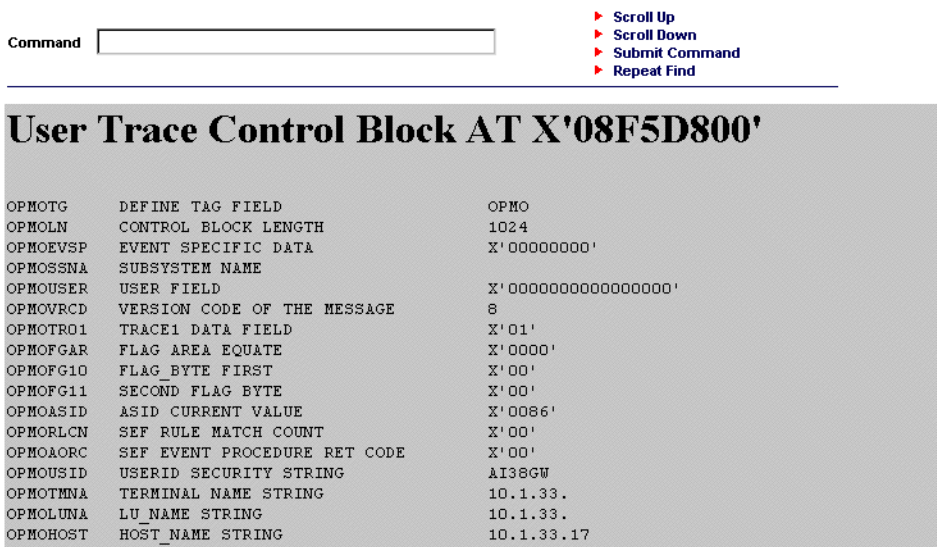
LOCATE	L	Scrolls the display a specific message. The LOCATE command has the following operands:
		• **date** - a date
		• **time** - a time in *hh:mm:ss* format
		• **date time** - a date time combination
		• **time date** - a time date combination
		• **msgno** - a specific message number. Messages are numbered beginning with 1 when SWS is started.
		Note: L 0 or L 1 will cause the trace browse display to begin the display with the top trace browse message that is available.
		Syntax examples:
		`L date`
		`L time`
		`L date time`
		`L time date`
		`L msgno`
		Examples:
		Date: “`L 3JUN99`”, or “`L 3JUN`”, or “`L 03JUN1999`”
		Time: “`L 11:`”, or “`L 11:00`”, or “`L 11:00:00`”
		Date Time: “`L 3JUN99 11:00:00`”
		Time Date: “`L 11:00:00 3JUN99`”
		General: “`L 1620`”

FIND	F	Searches for strings of characters within the trace list. Once found, the cursor is placed on the string. To find the string again, use the RFIND command.
		Syntax example:
		`FIND string column-name FIRST start-col end-col count`
		`  F      *              LAST`
		`                      PREV`
		`                        NEXT`
		Where:
		`string` is any character string. You must use quotes if there are imbedded blanks. You must use two quotes to imbed a quote mark in the search string. For example: **`f '''abc'''`** will find the string: **`'abc'`**

RFIND	RFIND	Repeat the FIND command (like RFIND in the ISPF editor). An RFIND following a FIRST find command will search forward (toward the latest messages), and an RFIND following a LAST command will search backwards (toward the oldest messages). **Syntax example:** `RFIND`

Zoom

The zoom features allows you to see the details on an individual record. For example:



User Trace Control Block AT X'08F5D800'

```
OPMOTG      DEFINE TAG FIELD                    OPMO
OPMOLN      CONTROL BLOCK LENGTH                1024
OPMOEVSP    EVENT SPECIFIC DATA                 X'00000000'
OPMOSSNA    SUBSYSTEM NAME
OPMOUSER    USER FIELD                          X'0000000000000000'
OPMOVRCD    VERSION CODE OF THE MESSAGE         8
OPMOTR01    TRACE1 DATA FIELD                   X'01'
OPMOFGAR    FLAG AREA EQUATE                    X'0000'
OPMOFG10    FLAG_BYTE FIRST                     X'00'
OPMOFG11    SECOND FLAG BYTE                    X'00'
OPMOASID    ASID CURRENT VALUE                  X'0086'
OPMORLCN    SEF RULE MATCH COUNT                X'00'
OPMOAORC    SEF EVENT PROCEDURE RET CODE        X'00'
OPMOUSID    USERID SECURITY STRING              AI38GW
OPMOTMNA    TERMINAL NAME STRING                10.1.33.
OPMOLUNA    LU_NAME STRING                      10.1.33.
OPMOHOST    HOST_NAME STRING                    10.1.33.17
```

Figure 16–12. Example of Using Zoom

This is an information only screen. You can search for information, scroll up or down, but you cannot edit the screen.

Trace Browse Control

When selected, each of the buttons across the top of the screen invoke a different panel.

Note

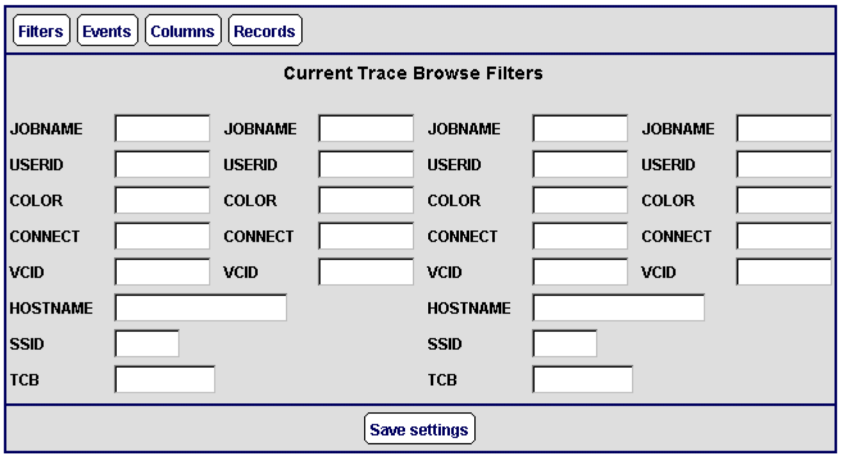
Each host has its own profile associated with it.

For example, if you have a copy of the Web Server with the hostid of `P390.NEONSYS.HOST1` and another with the hostid of `P390.NEONSYS.HOST2`, each hostid would have its own Shadow Web Interface trace browse profile specific to that host. Each profile could be configured individually.

This means, you could have a profile for HOST1 that shows the DATE and TIME columns, while the profile for HOST2 shows EVENT and TIME.

Filters Button

This panel will display your current settings and can be modified.



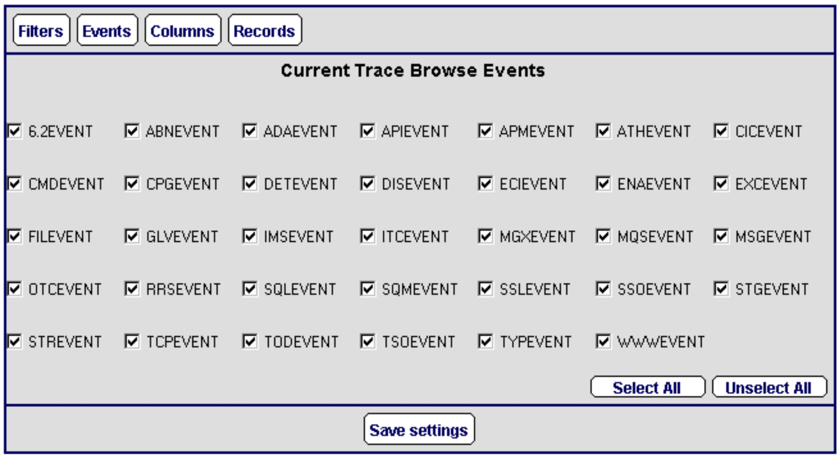
Current Trace Browse Filters

JOBNAME		JOBNAME		JOBNAME		JOBNAME	
USERID		USERID		USERID		USERID	
COLOR		COLOR		COLOR		COLOR	
CONNECT		CONNECT		CONNECT		CONNECT	
VCID		VCID		VCID		VCID	
HOSTNAME				HOSTNAME			
SSID				SSID			
TCB				TCB			

Figure 16–13. Example of the Filters Panel

Events Button

This panel displays the current events to monitor and can be modified.



Current Trace Browse Events

✓ 6.2EVENT	✓ ABNEVENT	✓ ADAEVENT	✓ APIEVENT	✓ APMEVENT	✓ ATHEVENT	✓ CICEVENT
✓ CMDEVENT	✓ CPGEVENT	✓ DETEVENT	✓ DISEVENT	✓ ECIEVENT	✓ ENAEVENT	✓ EXCEVENT
✓ FILEVENT	✓ GLVEVENT	✓ IMSEVENT	✓ ITCEVENT	✓ MGXEVENT	✓ MQSEVENT	✓ MSGEVENT
✓ OTCEVENT	✓ RRSEVENT	✓ SQLEVENT	✓ SQMEVENT	✓ SSLEVENT	✓ SSOEVENT	✓ STGEVENT
✓ STREVENT	✓ TCPEVENT	✓ TODEVENT	✓ TSOEVENT	✓ TYPEVENT	✓ WWWEVENT	

Figure 16–14. Example of the Trace Browse Events Panel

Columns Button

This panel displays the columns to monitor and can be modified.

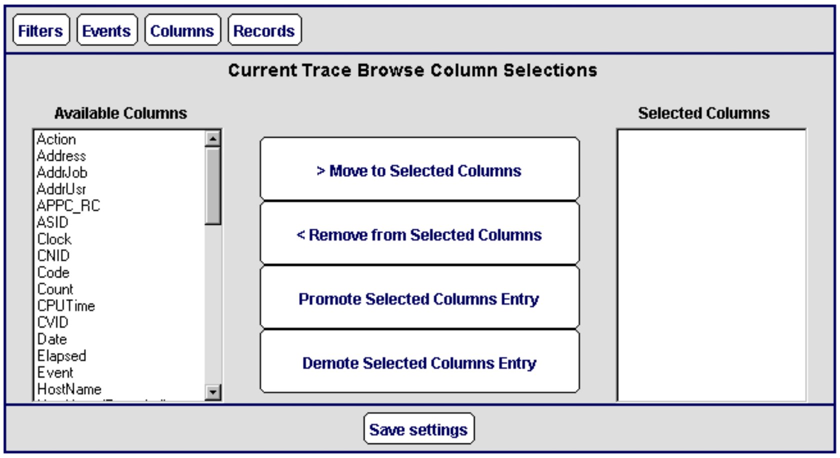
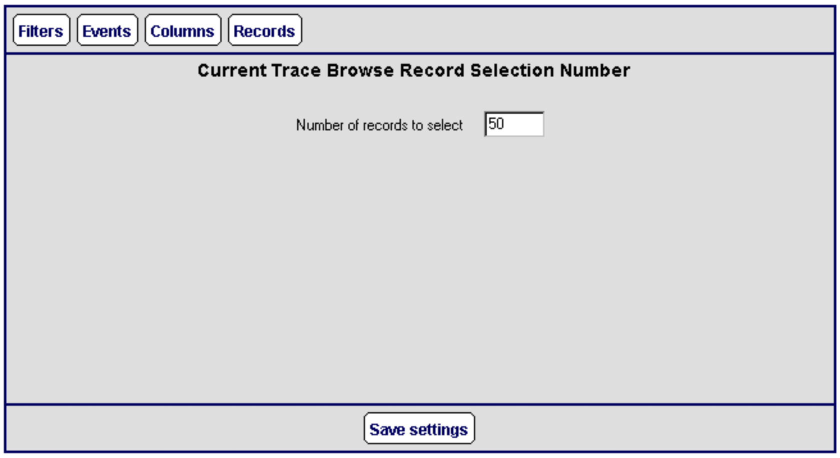


Figure 16–15. Trace Browse Columns

Records Button

This indicates the number of records to retrieve. The default is 50.



Number of records to select 50

Figure 16–16. Trace Browse Records

Security Features

The following security features are in Shadow Web Interface:

- **Encrypted userids and passwords.** Both are transmitted and stored in an encrypted format.
- **10 minute timeout.** There is an automatic 10 minute timeout that is invoked if you do not transmit a command via a mouse click. This security measure is active throughout the entire session. The timeout minutes *cannot* be reset.
- **Shadow OS/390 Web Server Security.** When you install the Shadow Web Server, additional security is invoked. Refer to the *Shadow Server Installation Guide* for more information on different security options.

CHAPTER 17: Using the OS/390 UNIX OpenEdition Hierarchical File System (HFS)

Shadow Web Server, Version 4.5.1 introduces support for the OS/390 Unix System Services Hierarchical File System (HFS). This new support:

- Allows you to use various off-the-shelf web-authoring tools to create and serve HTML and other application files using Shadow Web Server.
- Requires only minimal security subsystem configuration of OS/390 Unix System Services.

Steps to Setting up HFS

The following needs to be done before you can use HFS on your system:

1. Review OpenEdition security and implement any additional RACF, ACF/2, or TopSecret control parameters as needed.
2. Make sure the userid assigned to the Web Server started-task and the WWWDEFAULTRUNAUTH start-up parameter have an OMVS segment defined with a UID and GID value assigned.
3. Add the server start-up parameters to SWSxIN00.
4. Define the start-up ruleset definitions to SWSxIN00.
5. Create the rule(s).
6. Restart the system.
7. Use your Web browser and enter the URL.

OpenEdition and HFS Security

The OpenEdition Hierarchical File System handles security authorizations in a manner that is compatible with other UNIX operating systems, which is different than native MVS QSAM, VSAM, or PDS(E) datasets. If you are unfamiliar with the UNIX operating system, particularly file system security, we *strongly* suggest that you review the IBM *OpenEdition User's Guide* (SC28-1891) before proceeding. In particular, familiarize yourself with the concepts described in the "Handling Security for Your Files" section of this manual.

In order to secure your HFS-resident files properly, you may need to implement additional RACF, ACF/2, or TopSecret control parameters. We strongly recommend that you familiarize yourself with the OS/390 Unix System Services, and implement any new security administration policies and procedures, as appropriate.

OpenEdition Security Subsystem (RACF) Configuration

Before you can access HFS files using the Web Server, you must ensure that the MVS userid (UID) and group ID (GID) are defined with OMVS segments to RACF. Both IDs should have a value defined for OpenEdition.

Started-task Userid

The userid assigned to the Shadow Web Server started task must have an OMVS segment defined with a UID and GID value assigned. We recommend the UID value be unique.

Note:
A UID of zero (root) is *not* required by version 4.5.1 of the Web Server, *nor* is it recommended. The GID (group ID) should be different from the GID assigned to the default runtime userid.

This version of the Web Server retains it original native MVS organization and does not require that other special authorizations to any of the `BPX.*` resources be defined in the RACF `FACILITY` class. (Refer to the RACF documentation for more information.) Future versions of Shadow Web Server *may* require that the userid be granted authorization to various `BPX.*` resources.

Be sure that a real userid (not a default userid equal to asterisk (*)) is assigned to the Server’s started-task address space. If the SWSx started task userid is allowed to default, a userid might not be set up for the address space; this means, no UID or GID value will be assigned. You may need to implement support for the RACF `STARTED` class, or define the Server’s started-task name to RACF.

A UID/GID assignment is required when using OE Sockets with TCP/IP. Refer to the *Shadow Installation Guide* for more information.

Default Runtime Userid

The default userid, which is set by the `WWWDEFAULTRUNAUTH` start-up parameter and under which web transactions operate, must have an OMVS segment defined with a UID and GID value assigned. We strongly recommend that:

- Both the UID and GID values assigned to this userid be unique.
- Overall, they represent a relatively low level of authorization on the system.

The UID/GID assignment made for this userid must be authorized to perform OE Sockets TCP/IP operations. Refer to the *Shadow Installation Guide* for more information.

Important:
For *this* version of Shadow Web Server, all accesses to an HFS-resident file or directory path are performed under control of the authorizations granted to this userid and no other. The Server automatically switches the security environment to this userid before every access to an HFS-resident file, and then restores the pre-existing security environment afterwards.

The UID and/or GID associated with this userid must possess READ access to any files, and all traversed directory paths, that are to be served.

Server Start-up Parameters

In order to run HFS, you must enter the MODIFY PARM information into the SWSxIN00. By default, HFS support is not enabled. The following example shows one way to setup HFS:

```
IF 1 = 1 THEN DO                              /* ENABLE HFS? */
   "MODIFY PARM NAME(OEHFS) VALUE(ENABLE)"
   "MODIFY PARM NAME(HFSAUTHMODE) VALUE(GLOBAL)"
   "MODIFY PARM NAME(DOCUMENTROOT) VALUE(/u/)"
END                                       /* END HFS ACTIVATION        */
```

The parameters are explained in the following sections.

OEHFS Parameter (Required)

To globally enable support for the Hierarchical File System within the Server, you must specify the OEHFS start-up parameter value. Rules, which attempt access to an HFS file or path, cannot be enabled unless this parameter value is set during start-up.

Parameter Value	Usage
ENABLE	Specifies that HFS support is enabled. WWW rules that provide access to an HFS file or path are supported and can be enabled providing other operational controls are set.
DISABLE	Specifies the HFS support is disabled. WWW rules that attempt to provide access to an HFS file or path are not supported and cannot be enabled.

HFSAUTHMODE Parameter (Required)

The `HFSAUTHMODE` parameter determines how the Web Server handles HFS file security authorization processing when accessing and serving HFS-resident files. Currently, only one authorization operational mode (GLOBAL) is supported. Future versions *may* make other authorization processing options available.

Parameter Value	Usage
GLOBAL	Specifies that ALL accesses to the HFS be performed under the authorizations granted to the WWWDEFAULTRUNAUTH userid. Just before accessing any HFS file or path, the server switches the environment so that this userid’s authorizations are used. After retrieving the HFS file, the Server restores the existing security environment. This operational mode requires that the WWWDEFAULTRUNAUTH userid have read access to all HFS files and traversed directory paths that are to be made accessible via Shadow Web Server.

DOCUMENTROOT Parameter (Optional)

This parameter can be used to specify an HFS directory root path. Whenever a value is specified, it is used as a prefix for all **relative** pathnames in order to formulate the **absolute** pathname. It is NOT prepended to **absolute** pathnames.

Note:

An absolute path name begins with a leading slash (/), but a relative path name does not. The directory path for a relative reference must end with a slash (/). When `DOCUMENTROOT` is set to a NULL string, the value “/” is implied.

For example, a relative path would be, ‘uxy/’, while an absolute path would be, ‘/uxy/’.

To make ruleset-level and application-level promotions easier to manage, use the HFS directory prefix to differentiate between production and test copies of the Server. The value for this parameter:

- Is case sensitive.
- Must begin and end with a "/".
- Can be a NULL string.
- Is limited to 64 bytes in length.

Note:
ALL HFS pathname constructs are limited internally to a total of 256 bytes in length, which includes the length of this prefix, plus the ruleset-level `HFSROOT()` prefix (if non-NULL), plus the rule definition `PATH()` length, plus additional URL request bytes substituted for wildcards into the `PATH()` operand at execution time.

SEFV31COMPATIBLE Parameter

Existing customers that are using Version 3.1 compatible configuration to define SEF rulesets must first upgrade to use Version 4+ “`DEFINE RULESET`” configuration statements.

HFS access is not provided when the Server’s `SEFV31COMPATIBLE` start-up option is set to `YES`.

Ruleset Definitions

Before you can define/enable an individual `/*WWW` rule definition that provides access to HFS-resident files, you must first authorize the rules to appear within the encompassing SEF ruleset. To do this, code the `HFSROOT()` keyword in each designated ruleset to allow HFS-related `/*WWW` rule definitions. For example:

```
DEFINE RULESET NAME(WWW)
               RULETYPE(WWW)
               WWWCLASS(MASTER)
               HFSROOT(‘’)
               DSNAME(‘SHADOW.V451.WWW.MASTER.RULESET’)
```

The Web Server will not enable any HFS-related `/*WWW` rules that are defined within an SEF ruleset unless the `HFSROOT` keyword was coded for the ruleset’s definition at start-up.

HFSROOT()

The `HFSROOT()` operand of `DEFINE RULESET` specifies a relative or absolute pathname prefix string. This string is placed in front of the `PATH()` value for any `/*WWW` rule defined within the ruleset.

- **HFSROOT(NONE)** This explicitly disables HFS-related support within the corresponding ruleset. It is the only allowable value for all ruleset definitions except for WWW rulesets, which is assumed if not explicitly coded.
- **HFSROOT('')** The NULL string indicates no values are prefixed at the ruleset-level; yet, it allows HFS-related rule definitions within the ruleset. (A NULL string is processed as a relative pathname reference.)

For example:

```
"DEFINE RULESET NAME(NEON)"                          ,
                "RULETYPE(WWW) WWWCLASS(SUBORD)"    ,
                "HFSROOT('')"                        ,
                 "DSNAME('AMX232.SWSP.RULE.NEON.EXEC')"
```

- **HFSROOT(path)** The HFSROOT operand is intended to segregate HFS directory paths into the same departmental groups that can be managed using Shadow Web Server's ruleset-based architecture.

  Relative path example:

```
"DEFINE RULESET NAME(HFS)"                     ,
               "RULETYPE(WWW) WWWCLASS(SUBORD)"   ,
               "HFSROOT('abc123/')"       /* relative path */,
               "DSNAME('AMX232.SWSP.RULE.HFS.EXEC')"
```

  Absolute path example:

```
"DEFINE RULESET NAME(HFS)"                     ,
               "RULETYPE(WWW) WWWCLASS(SUBORD)"   ,
               "HFSROOT('/u/abc123/')"   /* absolute path */,
               "DSNAME('AMX232.SWSP.RULE.HFS.EXEC')"
```

HFSROOT vs. DOCUMENTROOT

Based on the values of HFSROOT and DOCUMENTROOT, one of the following will happen:

- When a **relative** pathname value is specified for HFSROOT, the Server's DOCUMENTROOT parameter is used as a prefix.
- When an **absolute** pathname value is used for HFSROOT, the DOCUMENTROOT parameter has no effect.

The HFSROOT pathname operand is case sensitive and may be from zero to 128 bytes in length.

/*WWW Rules

You provide mapping of in-bound request URLs to an HFS-resident file or path by coding a /*WWW rule definition using the PATH() keyword. Only the rule's header statement is coded; no process section (/*FILE) is permitted within such a rule definition.

For example:

```
/*www  /testrule/*  PATH('web/*')  WELCOMEPAGE('index.htm')
```

Where

- `/*www` is the rule’s header statement.
- `/testrule/*` is the rule.
- `PATH('web/*')` displays the subdirectory name, web.
- `WELCOMEPAGE('index.htm')` is the keyword and file name. (See “WELCOMEPAGE Keyword Operand” on page 17-7.)

The rule’s definition can include other keywords allowed for `/*WWW` statements, such as `AUTHREQ`, `RUNAUTH`, or `SENDTRACE or WELCOMEPAGE()`.

URL Criterion

The `/*WWW` rule’s URL matching criterion is specified the same as it is for all other WWW rule definitions. The criterion value may contain zero or one wildcard (*) character.

For example:

```
http://documentroot_value/subordinate_value/ruleset/index.htm
```

PATH Keyword Operand

The `PATH` operand:

- Consists of a string of 1-to-128 bytes in length and is case sensitive.
- Can contain zero or one wildcard character.
- Is not required to contain a wildcard, even if the URL matching criterion string contains one.

WELCOMEPAGE Keyword Operand

The `WELCOMEPAGE` operand is only valid when the `PATH` operand ends with a wildcard. This operand returns a specific file whenever an inbound request does not specify a file within a directory and the PATH operand specifies an HFS directory name.

Note:
If the `PATH` operand (preceding the wildcard character) does not designate a directory, the `WELCOMEPAGE` operand is merely appended to the value, and may generate an invalid (not found) reference.

Displaying the Web Page

Shadow Web Server uses the following to determine what is displayed in the Web browser:

```
DOCUMENTROOT + HFSROOT + PATH + WELCOMEPAGE
```

Note:
`DOCUMENTROOT` and `HFSROOT` are in the startup file.

`PATH` and `WELCOMEPAGE` are controlled by the rule.

APPENDIX A: *Trace Browse*

The trace browse application is used to view Shadow OS/390 Web Server’s trace records. The trace records contain information about communication and SQL processing events for all users (both attached and remote) of the system.

Starting Trace Browse

To start the trace browse application, select the trace browse option from the ISPF/SDF Primary Options Menu. The trace browse screen appears. It will look something like this:

```
--------------------  SHADOW SERVER BROWSE  - 08:27:53 31 MAR 92 COLS 001 064
COMMAND ===>                                                  SCROLL ===> PAGE
DDMMM HH:MM:SS ----+----1----+----2----+----3----+----4----+----5----+----6----
31MAR 08:27:53 CNOS FAILED - LU SDBIP00 - CNOS ALLOCATION FAILURE, RETRY
31MAR 08:27:53 DISPLAY FAILED - LU SDBIP00 - PARAMETER ERROR, NO CORRESPONDING
31MAR 08:27:53 CNOS FAILED - LU SDBTD01 - CNOS ALLOCATION FAILURE, RETRY
31MAR 08:27:53 DISPLAY FAILED - LU SDBTD01 - PARAMETER ERROR, NO CORRESPONDING
31MAR 08:28:42 CNOS FAILED - LU SDBIP00 - CNOS ALLOCATION FAILURE, RETRY
31MAR 08:28:42 DISPLAY FAILED - LU SDBIP00 - PARAMETER ERROR, NO CORRESPONDING
31MAR 08:28:42 CNOS FAILED - LU SDBTD01 - CNOS ALLOCATION FAILURE, RETRY
31MAR 08:28:42 DISPLAY FAILED - LU SDBTD01 - PARAMETER ERROR, NO CORRESPONDING
31MAR 08:30:06 CNOS FAILED - LU SDBIP00 - CNOS ALLOCATION FAILURE, RETRY
31MAR 08:30:06 DISPLAY FAILED - LU SDBIP00 - PARAMETER ERROR, NO CORRESPONDING
31MAR 08:30:06 CNOS FAILED - LU SDBTD01 - CNOS ALLOCATION FAILURE, RETRY
31MAR 08:30:06 DISPLAY FAILED - LU SDBTD01 - PARAMETER ERROR, NO CORRESPONDING
31MAR 08:43:47 EXTERNAL INTERRUPT       - PATH 0001 - NONPRIORITY MESSAGECOMPL
31MAR 08:43:47 SELECT EXECUTED          - PATH 0001 - SELECT COMPLETED
31MAR 08:43:57 SELECT STARTED           - PATH 0001 - SELECT INITIATED
31MAR 08:45:21 CNOS FAILED - LU SDBIP00 - CNOS ALLOCATION FAILURE, RETRY
31MAR 08:45:21 DISPLAY FAILED - LU SDBIP00 - PARAMETER ERROR, NO CORRESPONDING
31MAR 08:45:21 CNOS FAILED - LU SDBTD01 - CNOS ALLOCATION FAILURE, RETRY
31MAR 08:45:21 DISPLAY FAILED - LU SDBTD01 - PARAMETER ERROR, NO CORRESPONDING
***** ******** ************** BOTTOM OF MESSAGES *****************************
```

Figure A–1. Trace Browse Display

The trace browse application looks and functions very much like ISPF/PDF Browse. However, there are some important differences:

- On entry to trace browse, a message is displayed when a non-blank profile is being used.
- There is no dataset specification screen.

- The “dataset” displayed is the stream of trace messages that is constantly being extended at the bottom. When the maximum number of trace messages held is exceeded, messages at the top of the trace stream are removed. (The maximum number is set by the BROWSEMAX parameter.) The trace browse display will optionally reposition to the current bottom of the message stream each time you press <ENTER>.
- Supplemental information about the messages is available in columns that are displayed optionally (using the **DISPLAY** primary command).

Four-way scrolling is supported using the scroll commands (**UP**, **DOWN**, **LEFT**, **RIGHT**) or their associated PF keys.

Order of Trace Browse Events

As the server executes a particular SQL statement, several events are entered into the trace log on both the server and client sides. Both logs perceive the series of events from different perspectives, and each can have a different account of a singular event.

For instance, a client can execute a SQL statement and simultaneously enter the following events in its trace log.

```
SEND event
RECEIVE event
SQL event         (the results are returned)
```

The same three events will be logged on the server side as follows:

```
RECEIVE event     (matches the client SEND event)
SQL event         (the SQL statement is actually sent to DB2)
SEND event        (matches the client RECEIVE event)
```

The client side appears to be out of order until you consider that the sequences above are actually synchronized operations. If you could view a combined trace log, the SQL statement execution would appear as follows:

```
SEND event        (client side)
RECEIVE event     (server side)
SQL event         (server side)
SEND event        (server side)
RECEIVE event     (client side)
SQL event         (client side)
```

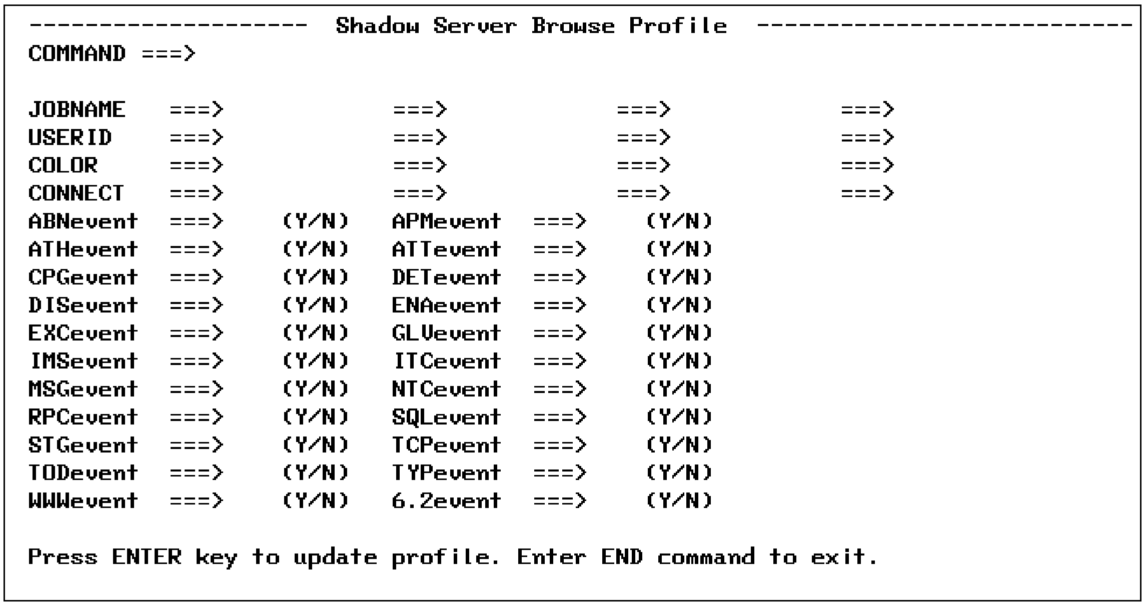
The Trace Browse Profile

You can view the server's events using the trace browse application. However, you may want to browse only a subset of these events. The trace browse profile can help you to do this by filtering the entire set of trace messages and only displaying those you want.The filtering profile is for an individual ISPF/PDF user. One user's profile has no affect on another user's.

When you first enter trace browse, you will have no profile and all messages will be displayed. To set a profile, you can use either the profile specification display or the trace browse profile command. The following sections discuss each method.

Using the Specification Display

To access the trace browse profile specification display, enter the **PROFILE** command, without any operands, on the command line of the main trace browse display. The profile specification display appears.



```
--------------------  Shadow Server Browse Profile  ---------------------------
COMMAND ===>

JOBNAME   ===>           ===>           ===>           ===>
USERID    ===>           ===>           ===>           ===>
COLOR     ===>           ===>           ===>           ===>
CONNECT   ===>           ===>           ===>           ===>
ABNevent  ===>    (Y/N)  APMevent  ===>    (Y/N)
ATHevent  ===>    (Y/N)  ATTevent  ===>    (Y/N)
CPGevent  ===>    (Y/N)  DETevent  ===>    (Y/N)
DISevent  ===>    (Y/N)  ENAevent  ===>    (Y/N)
EXCevent  ===>    (Y/N)  GLVevent  ===>    (Y/N)
IMSevent  ===>    (Y/N)  ITCevent  ===>    (Y/N)
MSGevent  ===>    (Y/N)  NTCevent  ===>    (Y/N)
RPCevent  ===>    (Y/N)  SQLevent  ===>    (Y/N)
STGevent  ===>    (Y/N)  TCPevent  ===>    (Y/N)
TODevent  ===>    (Y/N)  TYPevent  ===>    (Y/N)
WWWevent  ===>    (Y/N)  6.2event  ===>    (Y/N)

Press ENTER key to update profile. Enter END command to exit.
```

Figure A–2. Trace Browse Profile

The profile criteria that you specify in this panel determine which records are subsequently presented. For example, if you enter a jobname, only records having that jobname are displayed.

You can specify more than one profile criterion without conflict. Multiple entries are considered to be joined using the logical AND operator. Thus, if you enter two profile criteria, trace browse filters the records available so that only those that fit both criteria are displayed.

In the same way, you can specify more than one value for several profile criteria. When you specify more than one value for a profile field, the values are logically OR'd. For example, with two JOBNAMEs specified, a record will be selected if it contains one or the other of the values.

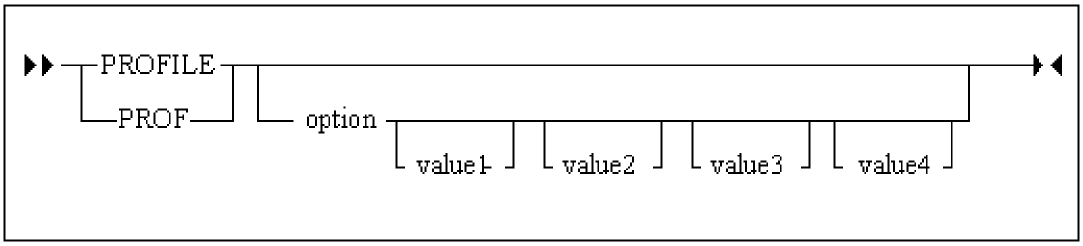
The following table describes the profile criteria (also called options) and the values that they may take.

Option	Limits Messages to Those That...	Value Description
JOBNAME	Are produced by this job.	Enter up to 4 values.
USERID	Have this userid.	Enter up to 4 values.
COLOR	(This option not supported.)	This option is not supported at this time
CONNECT	Have this connection ID.	Enter up to 4 values.
ATHevent	Are related to security authorization events.	Enter 'Y' for yes (default) or 'N' for no.
ATTevent	Are related to internal ATTACH events.	Enter 'Y' for yes (default) or 'N' for no.
ABNevent	Are related to internal ABEND events.	Enter 'Y' for yes (default) or 'N' for no.
IMSevent	(This option not supported.)	This option is not supported at this time
MSGevent	Are related to product initialization and termination messages.	Enter 'Y' for yes (default) or 'N' for no.
SQLevent	Are related to SQL statement execution.	Enter 'Y' for yes (default) or 'N' for no.
STGevent	Occur when a key change is required to update storage. These events occur when keys **0** through **7** are used or the key of storage.	Enter 'Y' for yes or 'N' for no (default).
TCPevent	Are related to TCP/IP communications.	Enter 'Y' for yes (default) or 'N' for no.
6.2event	Are related to LU 6.2 communications.	Enter 'Y' for yes (default) or 'N' for no.

Table A–1. Profile Criteria and Valid Values

Using the PROFILE Command

The **PROFILE** command can be used to either clear profile criteria or to establish new values. The syntax is:



option The name of the option you want to set.

Value1...value4
The values to use in selecting records by the option. Only JOBNAME, USERID, COLOR, and CONNECT can have multiple values. The “event” type options can only have one value (“Y” or “N”).

If you enter the **PROFILE** command without any operands, you will be presented with the profile specification display. Entering **PROFILE** option with the value omitted will clear the profile setting for that option and the option will not be considered for filtering.

The **PROFILE** option value format is for specifying profile criteria without using the profile specification display.

Examples

1. To enter a specification for `JOBNAME`, use:

   ```
   PROFILE JOBNAME AI38EDH
   ```

2. To select two jobs for trace browse, specify both with the same profile command:

   ```
   PROFILE JOBNAME AI38SCW AI38EDH
   ```

Wildcards for Trace Browse Profile

The `JOBNAME` and `USERID` criteria can contain wildcard specifications. A wildcard is an entry that ends with an asterisk (`*`). For example, if the entry in JOBNAME is ‘`AI38*`’, then all trace records jobnames which start with ‘`AI38`’ are selected.

Examples:

1. To filter out all Trace Browse messages except for those related to a particular connection, given that the connection ID is unique for each connection established with the product, use:

   ```
   PROFILE CONNECT connection-id
   ```

   The USERID and JOBNAME are unnecessary since a `connection-id` is more "granular" than either of these criteria. Use this type of profile whenever you want to study just one connection for a user. If you use the `connection-id` profile, you get all of the records for one session, including all communications, I/O, and SQL events.

2. To exclude all records except those produced by a single user, do the following:

   ```
   PROFILE JOBNAME jobname
   ```

   This shows you all the connections a user has made to SWS. You can use this type of profile whenever you are looking for patterns and need to study several sessions for a user.

Positioning Trace Browse

When first invoking trace browse, the display is positioned at the bottom of the list of trace records (you will see the "`Bottom of Messages`" marker at the bottom of the screen). Press <ENTER> to refresh the display with the latest messages. You can also refresh the display if the top of the trace browse list is displayed. (You will see the "`Top of Messages`" marker at the top of the screen). If the list is full, press the <ENTER> key to scroll the display downward. The older messages disappear to accommodate the newest messages being added to the end of the list.

If you reposition the trace browse display from its initial position at the bottom of the message stream it will no longer shift when you press the <ENTER> key. If you use the **DOWN MAX** command, the refresh mode will be reinstated (you will still need to press <ENTER> to see the latest messages).

Note:

Merely scrolling to the bottom without using the **DOWN MAX** command will not reinstate the refresh mode.

Changing Trace Browse Columns

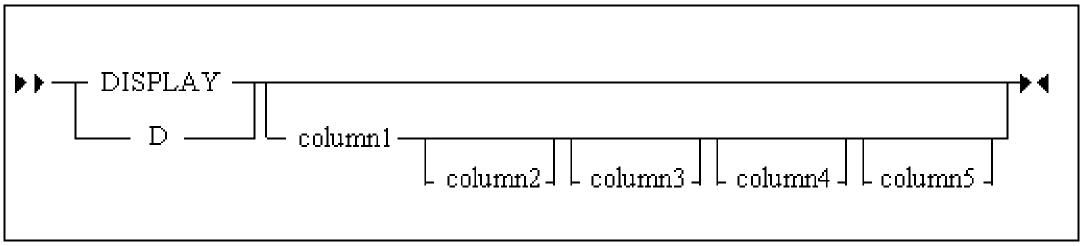
By default, trace browse displays three columns of information for each traced event:

- The time of the event
- The host name associated with the event
- A short description of the event

You can display many other columns as well with the **DISPLAY** command.

Displaying Extra Columns of Information

The format of the trace browse display can be changed using the **DISPLAY** command. The syntax is:



column1...column5
You can specify one to five display columns separated by blanks. The columns will appear to the left of the message text, and in the order that in which they are specified.

In order to clear the screen of the displayed columns, enter the **DISPLAY** command with no operands. The trace browse screen will display just the trace message text. (The trace message text is always included as a part of the trace browse no matter what other columns you specify).

Trace Browse Columns

The possible columns are:

Column	Description
ADDRESS	The message address. This is the location in memory of the actual message data. This column is used for product support and debugging.
ADDRJOB	The job address. This is the location in memory of the current entry in the jobname vector. This column is used for product support and debugging.

Table A–2. Trace Browse Columns

Column	Description
ADDRUSR	The userid address. The location in memory of the current entry in the userid vector. This column is used for product support and debugging.
ASID	The address space ID. The ASID of the address space that creates the current trace browse entry. This column is for general use and for product support.
CLOCK	The 8 byte binary clock value time stamp of when the trace browse message was created. This column is for general use and for product support.
CNID	The connection ID. The unique identifier assigned to each thread created by the product. This column is for general use and for product support.
COLOR	The color column. The color assigned to each trace browse message (very handy when using a monochrome monitor). This column is for general use and for product support. Note: The COLOR column is not completely implemented. At this time only the value NONE will be displayed.
CPUTIME	The CPU time used by a particular thread. The format depends on how much CPU time has used so far. If less than 1000 seconds has been used, the format is nnn.nnns. If more than 1000 seconds and less than 100 hours has been used, then the format is hh:mm:ss. If 100 hours or more has been used, the format is hhhhh:mm.
CVID	The conversation ID. LU 6.2 assigns this identifier when a conversation is started. This column is for general use and for product support.
CODE	The lowest level return code for each event in trace browse. This column is for general use and for product support.
DATE	The date in `dd:mm:yy` format when the message was created. This column is for general use and for product support.
ELAPSED	How long the current event took in decimal microseconds (millionths of a second). It is calculated by subtracting the STCK (store clock) value taken at the beginning of processing from the STCK value taken at the end of processing. This column is for general use and for product support.
EVENT	The type of event that created the message. The event types are as follows: • **6.2:** LU 6.2 event • **ABN:** ABEND event • **ATH:** Security authorization event • **ATT:** Internal attach event • **DET:** Internal detach event • **IMS:** IMS event • **MSG:** Trace message event • **NON:** No processing description • **SQL:** SQL event • **STG:** Storage event • **TCP:** TCP/IP event This column is for general use and for product support.
JOBID	The job ID of the job or address space that created the trace browse entry. This column is for general use and product support.
IPADDR	The Internet Protocol address. The TCP/IP source or target associated with the message. This column is for general use and product support.

Table A–2. Trace Browse Columns

Column	Description
JOBNAME	The jobname of the job or address space that created the trace browse entry. This column is for general use and product support.
LENGTH	The length of the text section of the message. This column is for general use and product support.
LUNAME	The LU 6.2 source or target associated with the message. This column is for general use and product support.
MSGNO	The message number. This is the sequential message number of the message. The first message collected by trace browse when data collection begins is message one. The second is message two, and so forth. When the capacity of the trace browse message area is exhausted, the oldest message is discarded as each new message is added. Because of this, the top message in trace browse is not necessarily message number one. This column is for general use and product support.
NODENAME	The node name. This is the communications node associated with the message. The format of each entry depends on the communication link type. This column is for general use and product support.
PATHID	The IUCV path ID. This is the path associated with the message. This column only has meaning for TCP/IP-related events. This column is for general use and product support.
RC	The highest level return code for the message. This column is for general use and for product support.
REASON	The reason code is the second level return code for the message. This column is for general use and product support.
SDBFLAGS	The product flags contain bits set by the various routines the create trace browse routines. This column is for product support and debugging.
SECONDS	The first 4 bytes of the binary time stamp of when the trace browse message was created. This column is for general use and product support.
SESSION	The session column shows the communications session associated with the message. The format of each entry depends on the communication link type. This column is for general use and product support.
SOCKET	The socket column shows the socket number associated with the message. This column only applies to TCP/IP-related events. This column is for general use and product support.
SQLRC	The SQL return code column contains the SQL return code associated with the message. This column applies to SQL-related events only. This column is for general use and for product support.
SUBSYSTEM	The subsystem column contains the DB2, IMS, or CICS subsystem name. This column is for general use and product support.
TCBADDR	The TCB address field contains the address of the TCB that created the message. This column is for general use and product support.
TCPRCEX	This column contains TCP/IP extended return codes. This column is only for TCP/IP related events. This column is for general use and product support.
TCPRC	This column contains TCP/IP return codes. This column is only for TCP/IP related events. This column is for general use and product support.
TERMNAME	The name of the terminal with which the event is associated.

Table A–2. Trace Browse Columns

Column	Description
TIME	The time, in hh:mm:ss format, when the message was created. This column is for general use and product support.
TIMEX	The extended time field. This is the time when the message was created calculated to the microsecond -- `hh:mm:ss.uuuuuu`
TRACE1	The trace data specific to the message. This field is for product support and debugging.
USERID	The security product userid that best identifies the message. This column is for general use and product support.
VERSION	The product version that created the message. This column is for general use and product support.
VTAMRC	This is the VTAM return code. This column is used mostly for product support.

Table A–2. Trace Browse Columns

Using Labels in the MSGNO Column

The MSGNO column is unique because it is the only modifiable column (you can type over the values in the column). In the MSGNO column, you can place “labels” which you can refer to using the **LOCATE** command. The format of trace browse labels is identical to the format of ISPF/PDF Edit labels:

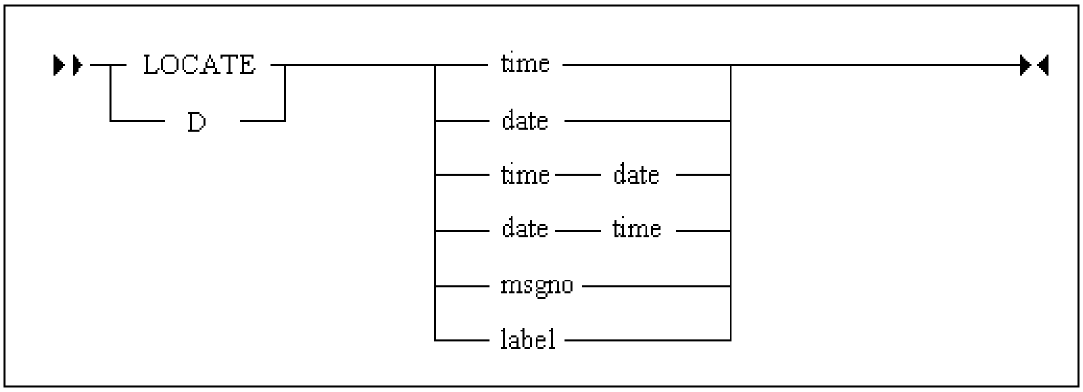
```
.aaaaaaa
```

A label consists of a period (.) followed by 1 to 7 alphabetic characters (a to z, upper or lowercase). Internally all label names are changed to uppercase for the purpose of comparison. Like ISPF/PDF Edit, you cannot use numbers in a label.

Locating Messages

Use the **LOCATE** command to position the display at a specific line. The line can be specified by date, time, date/time combination, or by message number.

The syntax is:



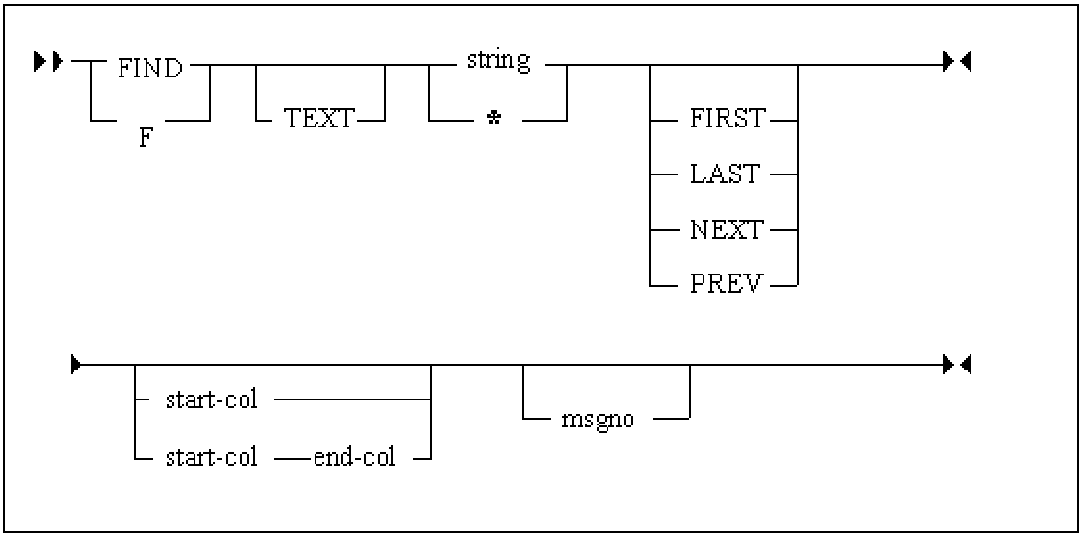
time	The time of day, in a 24-hour format, that you want the trace browse display scrolled. For example, 13:05:00 is 5 minutes past one in the afternoon. Use one of the following formats to specify the time: **hh:** hour only **hh:mm** hour and minute **hh:mm:ss** hour, minute, and second
date	The date that you want the trace browse display scrolled. Use one of the following formats to specify the date: **5APR** April 5, current year **05APR** April 5, current year **29FEB92** February 29, 1992 **29FEB1992** February 29, 1992
msgno	The message number to which you want the trace browse display scrolled. Message number is a 1 to 10 digit integer.
label	The name of a label which was previously set on a row in the `MSGNO` column. If the label name is not defined, an error message is displayed. The label name in the **LOCATE** command must begin with a period (.).

Using the FIND Command

Basically, the **FIND** command works like the ISPF/SDF **FIND** command; it is used to find character strings within the texts of messages. Trace Browse permits high speed finds against the columns of the **DISPLAY** command.

Finding Character Strings

The syntax of the **FIND** command is:



TEXT	An optional keyword indicating that the search is to take place against the text of the message and not against any other search columns.
string	The string to search for in the message text. If there are embedded blanks, or if the string is identical to a **FIND** keyword, it must be enclosed in quotes. Both single quotes and double quotes are accepted, with the restriction that a string must both begin and end with the same type of quote mark. If you want to include a quote mark within a string, you must "double-up" the quote marks. For example: `FIND 'this ain''t good english'` Alternatively, you can use one type of quote mark to delimit the string, and the other type as data within the string: `FIND "this ain't good english"`
*****	An asterisk indicates that the search string from the previous **FIND** command is to be used.
FIRST	Find the first occurrence of the string.
LAST	Find the last occurrence of the string.
PREV	Search upward.
NEXT	Search downward.
Start-col	Specifies the beginning text column for the search. Columns before `start-col` are not searched.
end-col	Specifies the ending text column for the search. Columns after `end-col` are not searched. If `start-col` is specified but `end-col` is not, `end-col` is assumed to be `start-col + length(string) - 1`.
Msgno	Specifies the number of messages to scan before abandoning the search. By default, 5000 messages are searched.

Note:
Trace browse is able to distinguish between `msgno` and `start-col` and `end-col` by examining the magnitude of the numbers. A number larger than 768 is assumed to be a msgno number and not a column number.

Repeating a FIND Command

To repeat a search, use the **RFIND** command. **RFIND** has no operands. It repeats the previous find command using the same search string, search direction, column limits, and message number limits. Normally, the **RFIND** command is assigned to <PF5>.

Finding With DISPLAY Columns

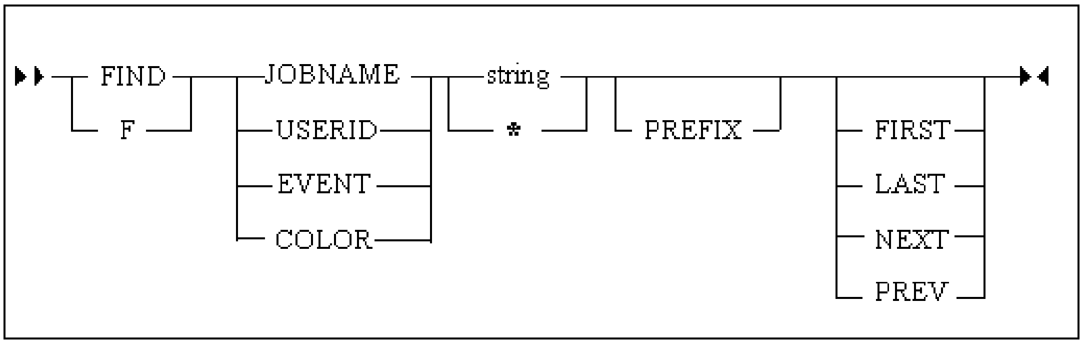
The **FIND** command can also be used to find information within some of the columns of the **DISPLAY** command. This search method is usually much faster than a text search.

When using the **FIND** command, be aware of the following:

1. There is no upper limit for searching columns. An unsuccessful search goes from the starting point to the end of the messages (for both upward and downward searches).

2. The **DISPLAY** column does not need to be visible for the column **FIND** to work. If the column is not visible, a successful search results in the cursor being placed in column 1 of the text field.

The syntax is:



JOBNAME
Specifies that the `JOBNAME DISPLAY` column is to be searched.

USERID Specifies that the `USERID DISPLAY` column is to be searched.

EVENT Specifies that the `EVENT DISPLAY` column is to be searched. The types of EVENTs that can be searched for are listed under "Displaying Extra Columns of Information" above.

COLOR Specifies that the `COLOR DISPLAY` column is to be searched.

string Search for this string in the message text. If there are embedded blanks, or if the string is identical to a **FIND** keyword, it must be enclosed in quotes. Both single quotes and double quotes are accepted, with the restriction that a string must both begin and end with the same type of quote mark. If you want to include a quote mark within a string, you must "double-up" the quote marks. For example:

```
FIND 'this ain''t good english'
```

Alternatively, you can use one type of quote mark to delimit the string, and the other type as data within the string:

```
FIND "this ain't good english"
```

***** An asterisk indicates that the search string from the previous **FIND** command is to be used.

PREFIX Specifies that the search string is a generic search string and gives only the prefix characters to be searched for. If you do not specify the PREFIX, keyword matching is byte-for-byte.

PREFIX is currently not supported for the `TEXT`, `COLOR`, and `EVENT` search columns.

FIRST Find the first occurrence of the string.

LAST Find the last occurrence of the string.

NEXT Search downward.

PREV Search upward.

Row Information Commands

There are four primary commands that can be used to invoke the special information displays for a particular trace browse row:

SWSZOOM
This is used to invoke the control block browse application, which presents formatted control block information for the selected row. This information is only used for product support. By default, <PF4> is set to execute the **SWSZOOM** command.

SWSINFO
This is used to invoke the SQL explanation application, which presents explanatory text regarding the SQLCODE associated with the selected row. By default, <PF6> is set to execute the **SWSINFO** command.

SWSTRAC
This is used to invoke the SQL trace application, which presents a trace of all SQL events for the connection ID associated with the selected row. By default, <PF16> is set to contain the **SWSTRAC** command.

SWSDATA
This is used to invoke the SQL data application, which presents a formatted SQL Communications Area (SQLCA) control block for the selected row. By default, <PF18> is set to contain the **SWSDATA** command.

These commands are used in conjunction with location of the cursor to determine which row to provide information for.

To invoke one of the special information displays, type the appropriate command in the command field, then position the cursor under the line in the display and press <ENTER>. Or place the cursor on the appropriate line and press the PF key associated with the command.

Printing Trace Browse Information

You can print any information from the trace browse application using the **P** and **PP** line commands.

- By entering the **P** line command into the `MSGNO` column, you will cause just the selected line to be printed.
- To print out a block of information, use the **PP** line command. Enter the command on both the first and last line of the block you want to print out. The **PP** command should be entered in the `MSGNO` column on your screen.

```
 --------------------  SHADOW SERVER TRACE --- 14:11:59 08 AUG 93  COLS 001 066
COMMAND ===>                                                 SCROLL ===> PAGE
MESG NUM EVN ----+----1----+----2----+----3----+----4----+----5----+----6----+-
00003629 TCP EXTERNAL INTERRUPT       - PATH 0001 - NONPRIORITY MESSAGE COMPLET
00003630 TCP GETHOSTNAME EXECUTED     - PATH 0001 - GETHOSTNAME COMPLETED
00003631 TCP EXTERNAL INTERRUPT       - PATH 0001 - NONPRIORITY MESSAGE COMPLET
PP       TCP SOCKET EXECUTED          - PATH 0001 SOCK 0003 - SOCKET COMPLETED
00003633 TCP EXTERNAL INTERRUPT       - PATH 0001 - NONPRIORITY MESSAGE COMPLET
00003634 TCP SETSOCKOPT EXECUTED      - PATH 0001 SOCK 0003 - SETSOCKOPT COMPLE
00003635 TCP EXTERNAL INTERRUPT       - PATH 0001 - NONPRIORITY MESSAGE COMPLET
00003636 TCP BIND EXECUTED            - PATH 0001 SOCK 0003 - BIND COMPLETED
00003637 TCP GETSOCKNAME STARTED      - PATH 0001 SOCK 0003 - GETSOCKNAME INITI
00003638 TCP EXTERNAL INTERRUPT       - PATH 0001 - NONPRIORITY MESSAGE COMPLET
PP       TCP GETSOCKNAME EXECUTED     - PATH 0001 SOCK 0003 - GETSOCKNAME COMPL
00003640 TCP EXTERNAL INTERRUPT       - PATH 0001 - NONPRIORITY MESSAGE COMPLET
00003641 TCP LISTEN EXECUTED          - PATH 0001 SOCK 0003 - LISTEN COMPLETED
00003642 TCP SELECT STARTED           - PATH 0001 - SELECT INITIATED
00003643 IMS CONTROL EXIT RESYNC COMMIT - REQUEST COMPLETED SUCCESSFULLY
00003669 ATH READ ACCESS TO NON.TRACEBROWSE ACCEPTED - RESOURCE SECURITY BYPASS
00003670 ATH READ ACCESS TO NON.TRACEBROWSE ACCEPTED - RESOURCE SECURITY BYPASS
******** *** **************** BOTTOM OF MESSAGES ******************************
```

Figure A–3. The PP Line Command

APPENDIX B: *Trace Browse Archival Facility*

The server's wrap-around trace facility is a powerful diagnostic tool designed to record critical events in the life of each individual transaction process. It is designed to record critical internal information which can be used to debug and correct problems within the server itself.

What is it

The wrap-around trace consists of a large block of virtual storage, which can optionally be backed by a data-in-virtual linear dataset. This block of virtual storage is sub-divided into a status area, a configurable number of event blocks, and a series of vector tables.

Status Area

The status area occupies the first 4k page of the trace virtual storage, and contains checkpoint information about the trace area, as well as information about the most recent trace archive.

Event Blocks

The event blocks begin within the second 4k page of the trace virtual storage area. The BROWSEMAX start-up parameter controls the number of event blocks allocated within this area. Each event block occupies 896 bytes of storage. Each server event is recorded into the next available slot, beginning with the first slot, continuing through the end of the event blocks, and then wrapping around to the beginning.

Vector Tables

Each vector table begins on a 4k page boundary which follows the event blocks in storage. The vector tables contain indexing information that allows views of the trace to be filtered without searching through the entire virtual storage area occupied by each individual event block.

How it works

When the trace is backed with a data-in-virtual dataset, it is checkpointed periodically to the dataset during server operations. The **BROWSEINTERVAL** start-up parameter sets the number of seconds between checkpointing operations.

When backed by a DIV dataset, the trace becomes persistent. This means that at each server restart, the wrap-around trace is continued from the point at which the last event was recorded before the previous product shutdown. Use of a DIV dataset also cuts down on the amount of virtual storage MVS must back within system page datasets, since checkpointed pages are paged out of virtual storage.

Each server event is recorded into the next event block within the wrap-around trace. The event records consist of a fixed length header and an event-specific recording area. For some event types, the recording area contains the actual text which you see when you view the trace. For other event types, binary information or internal control block images are placed into the recording area, but formatted as text when you view the records.

This configuration for the trace yields an extremely useful and powerful diagnostic tool while keeping the run-time overhead of supporting the facility at an absolute minimum. However, there are design tradeoffs inherent to this approach:

- The trace data-in-virtual dataset cannot be shared between two or more active servers.
- The event block slot locations, and size of the vector tables is fixed in relation to the total number of event block slots allocated. If you change the number of event slots later (that is, change the BROWSEMAX start-up parameter), the trace area must be reformatted. All pre-existing data will be lost.

Backups and Extracts

The server provides two facilities for creating copies of the active wrap-around trace. For archival purposes, the server can be configured to periodically make automatic backups of the trace. For diagnostic and reference purposes, the server allows user requested extracts to be made of the trace. These backups and extracts are recorded in exactly the same format as the active trace (each is a self-contained mini-trace).

This approach has several important benefits, plus a few drawbacks.

Benefits

- No detail is lost during backup processing that would otherwise occur if the records were extracted in text-only format.
- Extracted data remains in a very compact form and occupies no more DASD space than the original data.
- Extracted data can be reviewed almost instantly, because no heavy-weight pre-processing of the off-line logs into virtual storage is required.
- Data-in-virtual pages can be mapped instantly for review without scanning the data to re-create index information.

Drawbacks

- Each extract file is self-contained. This means individual backups cannot be merged together, since the sequencing of each event record and the indices which point to it are dependent on each event’s relative position within the DIV pages.

- The data remains in a proprietary format and cannot easily be processed by other utilities. However, because the data remains in a proprietary format, the server's configured security authorization controls cannot easily be circumvented.

Message Numbering

When each event is originally recorded within the wrap-around trace, it is assigned a sequential message number. Message numbering within a newly formatted trace begins at one, thereafter increments sequentially, and is continued during product restarts.

All backup and extract operations are performed using these message sequence numbers as a basis. For instance, server initiated automatic backup operations are scheduled based upon the number of new messages collected since the previous backup. User requested extracts are requested by specifying the beginning and ending message numbers to extract.

Configuring Automatic Backups

There are a number of server start-up parameters which you must set in order to configure automatic trace backup support.

1. Gain some experience with the active trace before attempting to configure automatic backup support. Specifically,

   a. Check to see how many trace events are being logged within a given time period and how frequently the trace wraps around.

   b. Adjust the BROWSEMAX parameter so that the active trace is sized appropriately.

   > ***Note:***
   > You want to avoid a thrashing condition, where the server is constantly building backups in order to record activity before the active trace wraps-around. Your active trace should be large enough to record at least a few hours activity before wrap-around occurs.

2. Set the BROWSEARCHIVECOUNT startup parameter for the number of messages to be copied each time an automatic backup operation is scheduled. This count should normally be 20% to 80% of the BROWSEMAX value. Set the count value high enough so backup operations are not constantly underway, but low enough that even under heavy load, the active trace will not wrap around before activity can be backed up.

   During this time, the Server does not suspend operation, nor does it stop recording new events, even if the new activity begins to overlay messages that have not been backed up.

3. Set the BROWSEARCHIVECUSHION start-up parameter for the number of messages to be used as a scheduling threshold or cushion for backup operations. This cushion value is used by the server to avoid creating archives in which some messages have been overlaid due to trace wrap-around.

4. Set data-in-virtual dataset allocation parameters. These parameters include:

   - ARCHIVEDSNPREFIX
   - ARCHIVEDATACLASS
   - ARCHIVEMGMTCLASS
   - ARCHIVESTORCLASS
   - ARCHIVEDEFCLPARMS

5. Set the BROWSEARCHIVE parameter to “AUTO” to activate automatic backup processing.

To Begin Testing

Normally, we suggest that you begin testing the automatic backup facility by setting the BROWSEARCHIVECOUNT parameter to 30% of your BROWSEMAX count, and the BROWSEARCHIVECUSHION to 50% of the BROWSEARCHIVECOUNT value.

The server schedules automatic backup operations using these configured values. However, if it detects that the values are inappropriate, it will override these values during start-up.

Data Set Allocation Parameters

The following table contains the data set allocation parameters:

Parameter	Description
ARCHIVEDSNPREFIX	Defines the high-level qualifier used by the sub-system to construct datasets name for trace browse archive files. The value “`.Dyyyyddd.Thhmmss`” is appended to the qualifier, where “`yyyyddd`” is the Julian date, and “`hhmmss`” is the time of day. Trace browse archival processing cannot be performed if this prefix is not set, because there is no default value.
ARCHIVEDATACLASS	Defines the DATACLASS operand value used to define linear clusters for archive datasets. If it is not set, DATACLASS is not specified when the linear datasets are allocated.
ARCHIVEMGMTCLASS	Defines the MGMTCLASS operand value used to define linear clusters for archive datasets. If it is not set, MGMTCLASS is not specified when the linear datasets are allocated.
ARCHIVESTORCLASS	Defines the STORCLASS operand value used to define linear clusters for archive datasets. If is not set, STORCLASS is not specified when the linear datasets are allocated.

Table B–1. Data Set Allocation Parameters for Trace Browse Archive

Parameter	Description
ARCHIVEDEFCLPARMS	Contains additional parameter values which are passed on DEFINE CLUSTER statements generated to define archive backup datasets.

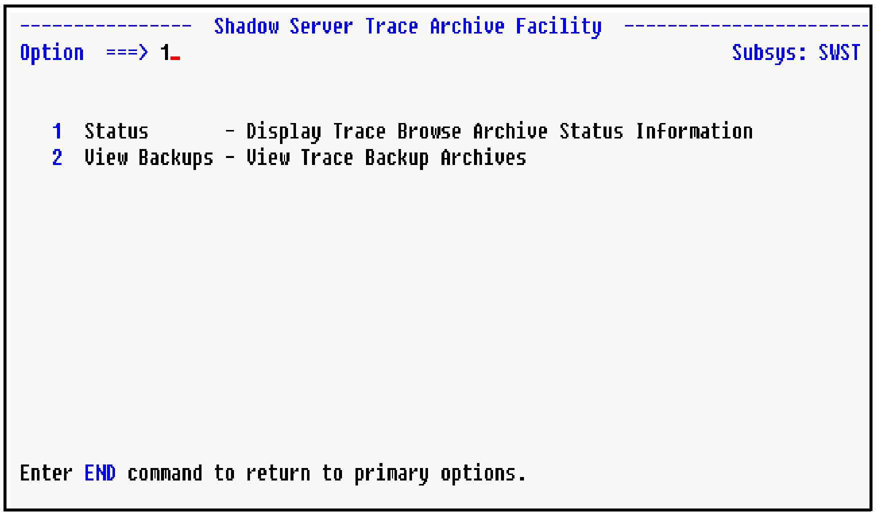
Table B–1. Data Set Allocation Parameters for Trace Browse Archive

Using the Trace Browse Archival Facility

To access the Trace Browse Archival Facility:

1. Select SWS Control from the Primary Options Menu.
2. Select Trace Archive.

The following panel appears:



```
----------------  Shadow Server Trace Archive Facility  -----------------------
Option  ===> 1_                                                   Subsys: SWST


   1  Status       - Display Trace Browse Archive Status Information
   2  View Backups - View Trace Backup Archives











Enter END command to return to primary options.
```

Figure B–1. Trace Archive Facility

From this panel you can select the following options:

Status Use this to view the trace browse archive status information shown in Figure B–2, Figure B–3, Figure B–4, and Figure B–5. These panels give you complete backup/archive status information including control values, automatic backup control parameters, backup dataset allocation parameters, and a sample of an IDCAMS statement generated by the server for archive backup dataset allocations.

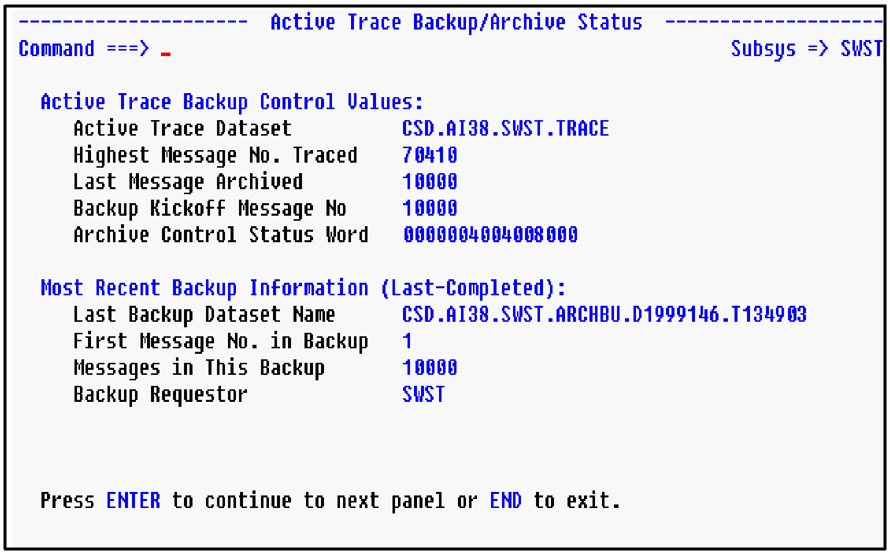
View Backups
Use this to view trace archive backups. After selecting the dataset name to view, you can view all the backup trace archives for that dataset. See Figure B–6 and Figure B–7.

The following sections take you through these panels step-by-step.

Note:
These panels are view-only.

To view active trace backup/archive status:

1. Select **Status** from the Trace Archive Facility panel (shown in Figure B–1). The following panel appears:

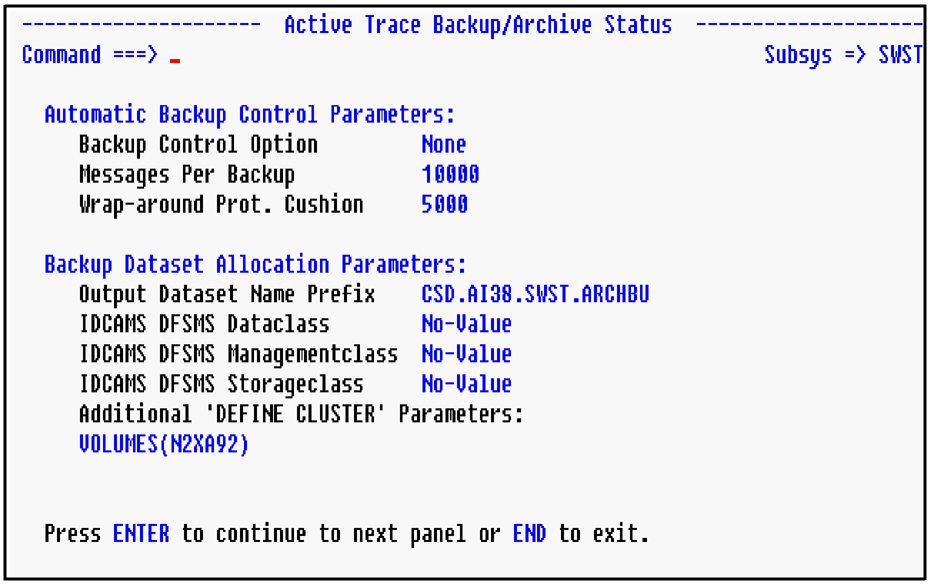


```
--------------------  Active Trace Backup/Archive Status  -------------------
Command ===> _                                                    Subsys => SWST

  Active Trace Backup Control Values:
     Active Trace Dataset           CSD.AI38.SWST.TRACE
     Highest Message No. Traced     70410
     Last Message Archived          10000
     Backup Kickoff Message No      10000
     Archive Control Status Word    0000004004008000

  Most Recent Backup Information (Last-Completed):
     Last Backup Dataset Name       CSD.AI38.SWST.ARCHBU.D1999146.T134903
     First Message No. in Backup    1
     Messages in This Backup        10000
     Backup Requestor               SWST



  Press ENTER to continue to next panel or END to exit.
```

Figure B–2. Backup/Archive Status, Panel 1

   This panel display the name of the active trace dataset, and the latest message information. The second half of the panel gives you status information for the most recent backup. The subsystem name is located on the upper right hand corner of the panel.

2. Press <Enter> to go to the second status panel. (See Figure B–3.)

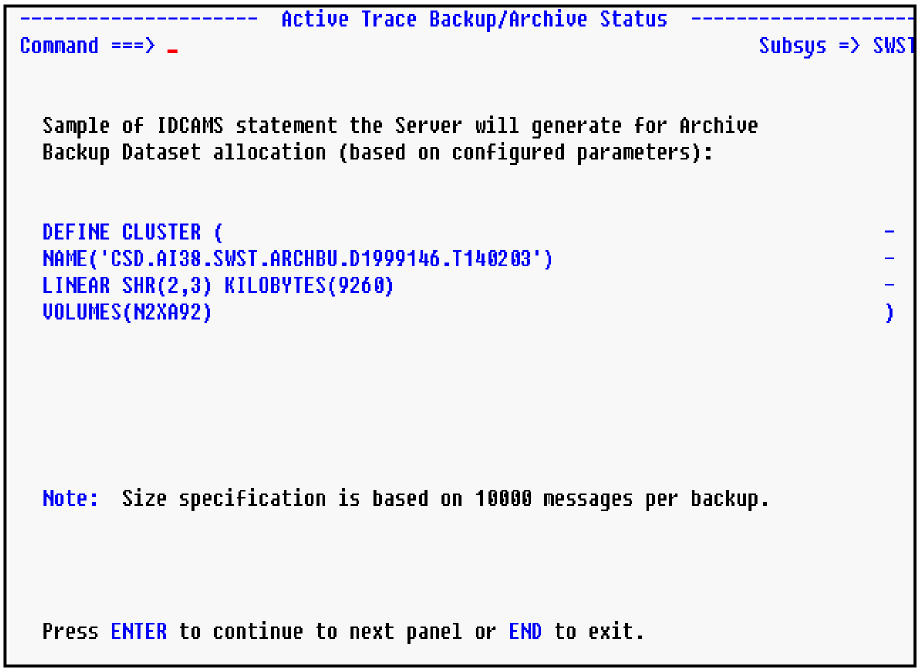


```
---------------------  Active Trace Backup/Archive Status  --------------------
Command ===> _                                                     Subsys => SWST

  Automatic Backup Control Parameters:
     Backup Control Option          None
     Messages Per Backup            10000
     Wrap-around Prot. Cushion      5000

  Backup Dataset Allocation Parameters:
     Output Dataset Name Prefix     CSD.AI38.SWST.ARCHBU
     IDCAMS DFSMS Dataclass         No-Value
     IDCAMS DFSMS Managementclass   No-Value
     IDCAMS DFSMS Storageclass      No-Value
     Additional 'DEFINE CLUSTER' Parameters:
     VOLUMES(N2XA92)


  Press ENTER to continue to next panel or END to exit.
```

Figure B–3. Backup/Archive Status, Panel 2

This panel shows the parameter settings for automatic backup control and for backup dataset allocation.

3. Press <Enter> to go to the next status panel. (See Figure B–4.)

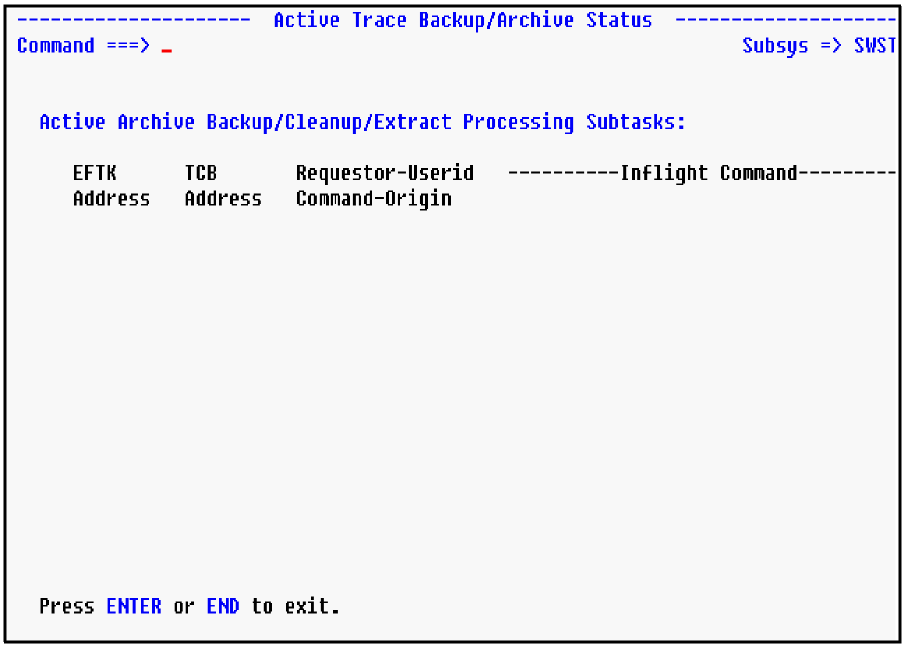


```
---------------------  Active Trace Backup/Archive Status  --------------------
Command ===> _                                                     Subsys => SWST


  Sample of IDCAMS statement the Server will generate for Archive
  Backup Dataset allocation (based on configured parameters):


  DEFINE CLUSTER (                                                        -
  NAME('CSD.AI38.SWST.ARCHBU.D1999146.T140203')                           -
  LINEAR SHR(2,3) KILOBYTES(9260)                                         -
  VOLUMES(N2XA92)                                                         )





  Note:  Size specification is based on 10000 messages per backup.




  Press ENTER to continue to next panel or END to exit.
```

Figure B–4. Backup/Archive Status, Panel 3

This panel gives a sample of the IDCAMS statement that will be generated by the server for the backup dataset allocation. It is based on the parameter values shown in Panel 2.

4. Press <Enter> to go to the final panel for the Status option. (See Figure B–5.)



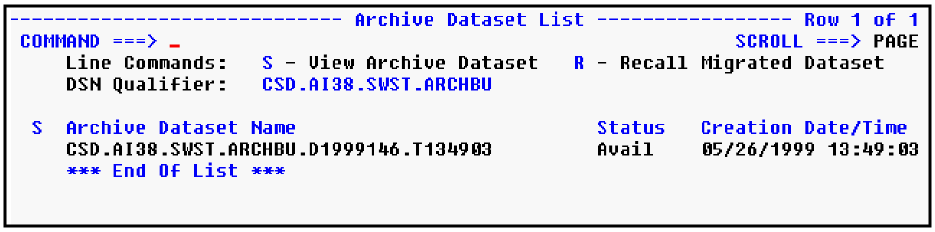
```
---------------------  Active Trace Backup/Archive Status  --------------------
Command ===> _                                                    Subsys => SWST


  Active Archive Backup/Cleanup/Extract Processing Subtasks:

     EFTK      TCB       Requestor-Userid   ----------Inflight Command---------
     Address   Address   Command-Origin

















  Press ENTER or END to exit.
```

Figure B–5. Backup/Archive Status, Panel 4

This panel displays the subtask information for archive backup/cleanup/ extract processing.

To view trace backup archives:

1. Select **View Backups** from the Trace Archive Facility panel (shown in Figure B–1) to access the Archive Dataset List panel:

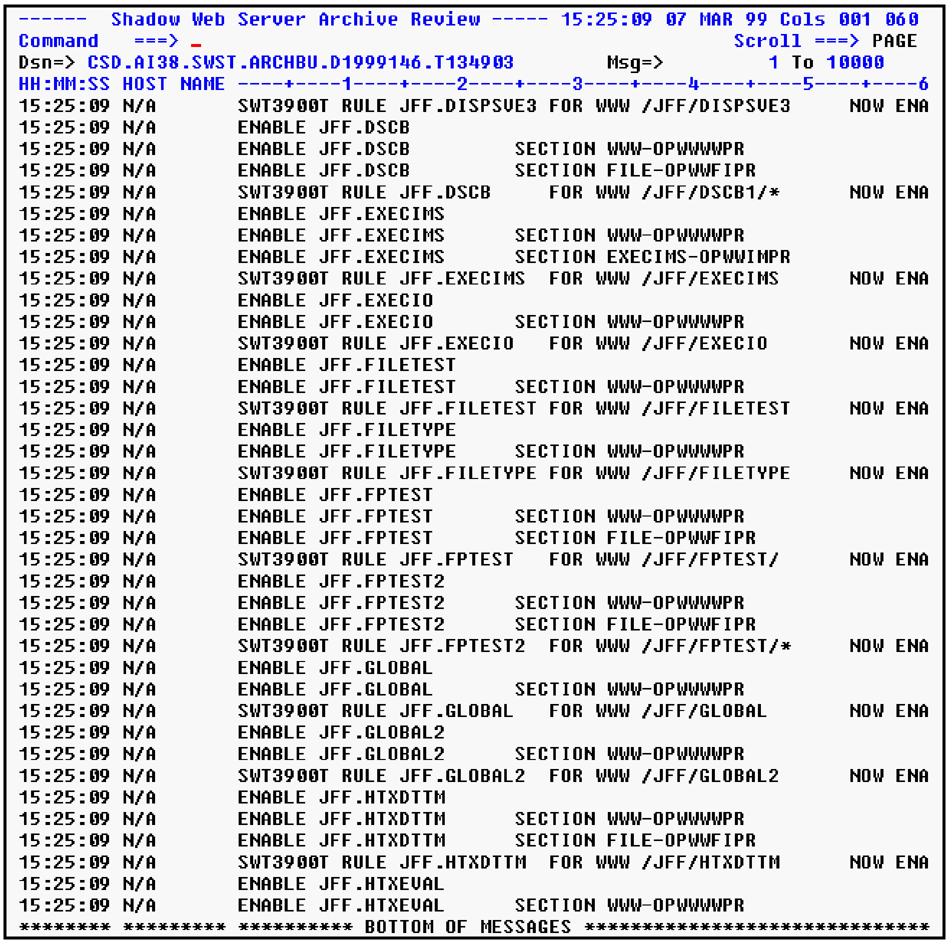


```
----------------------------- Archive Dataset List ----------------- Row 1 of 1
COMMAND ===> _                                              SCROLL ===> PAGE
    Line Commands:   S - View Archive Dataset   R - Recall Migrated Dataset
    DSN Qualifier:   CSD.AI38.SWST.ARCHBU

 S  Archive Dataset Name                         Status   Creation Date/Time
    CSD.AI38.SWST.ARCHBU.D1999146.T134903        Avail    05/26/1999 13:49:03
    *** End Of List ***
```

Figure B–6. Archive Dataset List Panel

This panel displays the active dataset name for which backup will be displayed.

2. Type *S* next to the archive dataset name and press <Enter>. This takes you to the next panel which displays the ttrace archive backup for that dataset (see Figure B–7):



```
------  Shadow Web Server Archive Review ----- 15:25:09 07 MAR 99 Cols 001 060
Command   ===> _                                            Scroll ===> PAGE
Dsn=> CSD.AI38.SWST.ARCHBU.D1999146.T134903        Msg=>        1 To 10000
HH:MM:SS HOST NAME ----+----1----+----2----+----3----+----4----+----5----+----6
15:25:09 N/A       SWT3900T RULE JFF.DISPSVE3 FOR WWW /JFF/DISPSVE3     NOW ENA
15:25:09 N/A       ENABLE JFF.DSCB
15:25:09 N/A       ENABLE JFF.DSCB         SECTION WWW-OPWWWWPR
15:25:09 N/A       ENABLE JFF.DSCB         SECTION FILE-OPWWFIPR
15:25:09 N/A       SWT3900T RULE JFF.DSCB     FOR WWW /JFF/DSCB1/*      NOW ENA
15:25:09 N/A       ENABLE JFF.EXECIMS
15:25:09 N/A       ENABLE JFF.EXECIMS      SECTION WWW-OPWWWWPR
15:25:09 N/A       ENABLE JFF.EXECIMS      SECTION EXECIMS-OPWWIMPR
15:25:09 N/A       SWT3900T RULE JFF.EXECIMS  FOR WWW /JFF/EXECIMS      NOW ENA
15:25:09 N/A       ENABLE JFF.EXECIO
15:25:09 N/A       ENABLE JFF.EXECIO       SECTION WWW-OPWWWWPR
15:25:09 N/A       SWT3900T RULE JFF.EXECIO   FOR WWW /JFF/EXECIO       NOW ENA
15:25:09 N/A       ENABLE JFF.FILETEST
15:25:09 N/A       ENABLE JFF.FILETEST     SECTION WWW-OPWWWWPR
15:25:09 N/A       SWT3900T RULE JFF.FILETEST FOR WWW /JFF/FILETEST     NOW ENA
15:25:09 N/A       ENABLE JFF.FILETYPE
15:25:09 N/A       ENABLE JFF.FILETYPE     SECTION WWW-OPWWWWPR
15:25:09 N/A       SWT3900T RULE JFF.FILETYPE FOR WWW /JFF/FILETYPE     NOW ENA
15:25:09 N/A       ENABLE JFF.FPTEST
15:25:09 N/A       ENABLE JFF.FPTEST       SECTION WWW-OPWWWWPR
15:25:09 N/A       ENABLE JFF.FPTEST       SECTION FILE-OPWWFIPR
15:25:09 N/A       SWT3900T RULE JFF.FPTEST   FOR WWW /JFF/FPTEST/      NOW ENA
15:25:09 N/A       ENABLE JFF.FPTEST2
15:25:09 N/A       ENABLE JFF.FPTEST2      SECTION WWW-OPWWWWPR
15:25:09 N/A       ENABLE JFF.FPTEST2      SECTION FILE-OPWWFIPR
15:25:09 N/A       SWT3900T RULE JFF.FPTEST2  FOR WWW /JFF/FPTEST/*     NOW ENA
15:25:09 N/A       ENABLE JFF.GLOBAL
15:25:09 N/A       ENABLE JFF.GLOBAL       SECTION WWW-OPWWWWPR
15:25:09 N/A       SWT3900T RULE JFF.GLOBAL   FOR WWW /JFF/GLOBAL       NOW ENA
15:25:09 N/A       ENABLE JFF.GLOBAL2
15:25:09 N/A       ENABLE JFF.GLOBAL2      SECTION WWW-OPWWWWPR
15:25:09 N/A       SWT3900T RULE JFF.GLOBAL2  FOR WWW /JFF/GLOBAL2      NOW ENA
15:25:09 N/A       ENABLE JFF.HTXDTTM
15:25:09 N/A       ENABLE JFF.HTXDTTM      SECTION WWW-OPWWWWPR
15:25:09 N/A       ENABLE JFF.HTXDTTM      SECTION FILE-OPWWFIPR
15:25:09 N/A       SWT3900T RULE JFF.HTXDTTM  FOR WWW /JFF/HTXDTTM      NOW ENA
15:25:09 N/A       ENABLE JFF.HTXEVAL
15:25:09 N/A       ENABLE JFF.HTXEVAL      SECTION WWW-OPWWWWPR
******** ********* ********** BOTTOM OF MESSAGES ******************************
```

Figure B–7. Archive Review

APPENDIX C: *Starting a Test Version*

Shadow OS/390 Web Server's debugging control allows you start and stop test copies of Shadow OS/390 Web Server under your TSO session. Running a test copy allows you to use all of the standard debugging tools to help you solve problems that can arise. After you finish writing the application, transfer them to the library of the real Shadow OS/390 Web Server. They should operate perfectly.

Setting Up Shadow Server to Run under TSO

Before you can run Shadow OS/390 Web Server under a TSO user's address space, the TSO user must be set up to run exactly as the Server. For setup information, refer to the appendix in the *Installation Guide*.

Test Copies

Test copies of Shadow OS/390 Web Server supports multiple users and operates similarly to the real server. The only differences are:

- You have access to all applications and sub-applications available with Shadow OS/390 Web Server. There is be no security checking.
- Performance is not as high. The product does not operate as quickly in a TSO session.

Note:
Running Shadow OS/390 Web Server under TSO is not supported with the TSOPLUS product. If you have TSOPLUS installed, you must setup another TSO logon proc that uses the standard `IKJEFT01` program.

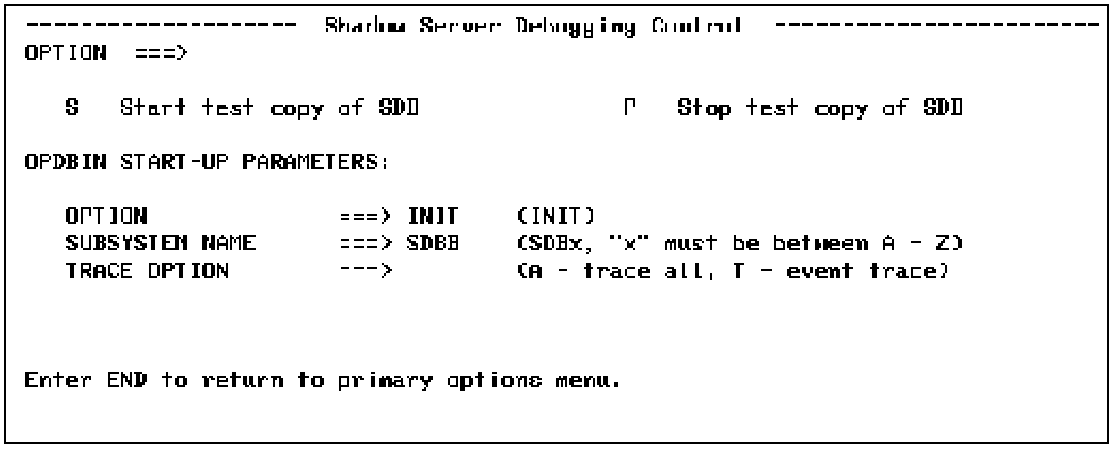
In order to run Shadow OS/390 Web Server under a TSO user's address space, the TSO user needs to be set up to run exactly as the server. For details on any of the following steps, please refer to the *Installation Guide*.

Using the Debugging Control Screen

The debugging screen specify option `S` to start a test copy, and option `P` to stop a test copy. You can specify the following parameters:

Parameter	Description
OPTION	Specify INIT. This is the processing option for OPDBIN. It causes an SDB main address space to be initialized.
SUBSYSTEM NAME	Specify the MVS 4-character subsystem name in the same way that you would with a real copy of Shadow OS/390 Web Server.
TRACE OPTION	• Specify `A` if you want all communication events to be traced with full control block dumps. • Specify `T` if you want all communication events to be traced with one line messages.

Table C–1. Debugging Parameters



```
-------------------  Shadow Server Debugging Control  ----------------------
OPTION  ===>

   S   Start test copy of SDB              P   Stop test copy of SDB

OPDBIN START-UP PARAMETERS:

   OPTION              ===> INIT    (INIT)
   SUBSYSTEM NAME      ===> SDBB    (SDBx, "x" must be between A - Z)
   TRACE OPTION        --->         (A - trace all, T - event trace)



Enter END to return to primary options menu.
```

Figure C–1. The Server Debugging Control Panel

Using the Code/370 Debug Tool

1. Compile the program using LE370 with the `TEST` option in both the compile and link steps.
2. Use the Shadow OS/390 Web Server's ISPF panels, go to the debug panel to start the debug subsystem (you must run a debug copy of Shadow OS/390 Web Server in your TSO address space to use this option).
3. Press <ENTER> until a message is received that your subsystem has started.
4. Go to option 5.
5. Enter the Language (such as, COBOL), enter `TEST` for the run-time option, and enter any required parameters.
6. The terminal locks once the RPC is running.

7. Connect to the Shadow OS/390 Web Server subsystem using any tool that can invoke the RPC and the debug screen will be displayed

The use of the debugger is documented in the IBM manual *IBM Debug Tool - User's Guide and Reference*.

APPENDIX D: Sever Error Codes

The following list of Server Error Codes describes various error conditions encountered during the processing of a URL. When combined with the Server Error HTML page and viewed in the context of the task being performed, they provide a better understanding of the error incurred.

Code	Description
1	Method parse failed
2	URL string parse failed
3	Variable initialization error
4	HTTP header line invalid
5	Authorization scheme invalid
6	Authorization routine failed
7	URL unmatched and Rescan is disabled
8	Shadow Event Facility (SEF) failed
9	Variable defaults routine failed
10	Receive timed out or failed
11	Zero bytes received
12	Header not terminated with blank line
13	Method unsupported
14	Method undefined
15	URL size greater than max criterion
16	Too many or too few separators
17	Parse of query data failed
18	Storage getmain failed
19	WWEV not present for SMF
20	SEF Rule processing abended
21	All transactions must be authorized
22	Compiler or Interpretor Failed
23	System Error procedure failed
24	Infinite pattern match loop

Table D–1. Server Error Codes

Code	Description
25	AUTHREQ or RUNAUTH check failed
26	URL resource refused
27	Invalid userid in RUNAUTH
28	Logoff routine failed
29	Auxillary component failed
30	PDS(E) member not found
31	DDName or dataset name not found
32	content_type: not set
33	Rule does not contain inline data
34	Session failure during transmit
35	Rescan URL not found
36	Not authorized for dataset
37	WORKSPACE parameter override failed
38	QUEUESPACE parameter override failed
39	Failure sending INPUTFORM
40	Secured Sockets Layer (SSL) processing failed
41	SSL connection required
42	No carriage return/line feed (CRLF) pair for HTTP header
43	Boundary marker parse errors
44	Multi-part or Form-data errors
45	File too large
46	User program not found in RPCLIB
47	Datamap not found
48	No data fields in input datamap
49	Input buffer too large (greater than 32704)
50	Transaction code nor Command in input message
51	APPC/MVS connection failed
52	/FOR IMS Command requires output map (MOD) name
53	Missing HTML variable (SWSINMAP, SWSNXTRN, PFKIN and/or SWSCNVID)
54	IMS command not supported
55	ABENDED detected while processing a dynamic (embedded) rule

Table D–1. Server Error Codes

Code	Description
56	HFS file is not found
57	HFS path is invalid
58	HFS path is too large
59	Dataset was migrated/offline
60	Exit-Abort was coded in the HTX file
61	HTX rescan URL is invalid
62	HTX embedded rules are disabled
63	HFS pathname cannot be correctly resolved because it contains an invalid or unauthorized combination of ./ or ../ characters.
101 through 199	These represent REXX evaluation errors. The value 100, is added to the standard <a href="shadrx1.htm#Complier Error Messages">REXX-language return code value</a>. (For example: 148 = REXX-Language error 48 = "FAILURE IN SYSTEM SERVICE").

Table D–1. Server Error Codes

APPENDIX E: *Supported SMF Fields*

To enable Shadow OS/390 Web Server’s SMF recording, set the SMFNUMBER parameter to the number you want. If the parameter is set to zero (0), no logging takes place. See the "Customize Initialization EXEC" section in the *Installation Guide* for more information.

SMF Type 05 Records

These records are written out by Shadow OS/390 Web Server each time a URL is executed. The layout for the 05 records can be found in member OPSMRC of the `‘NEON.xxxxxx.ASM’`* dataset.

A sample SAS program has been provided which can be used to print out these SMF fields. The program is located in the `‘NEON.SVxxxxxx.CNTL’` dataset, member SMFSWS05.

Offset	Field Name	Field Type or Value	Description
0	SMFHLN	BL2	RECORD LENGTH
2	SMFHSG	BL2	SEGMENT DESCRIPTOR
4	SMFHFG	BL1	HEADER FLAG BYTE • `x’10’` = MVS/ESA 4 • `x’08’` = MVS/XA • `x’04’` = MVS/ESA • `x’02’` = VS2
5	SMFHRCTY	BL1	RECORD TYPE
6	SMFHTIME	BL4	RECORD WRITTEN TIME (TIME BIN)
10	SMFHDATE	PL4	RECORD WRITTEN DATE (0CYYDDDF)
14	SMFHSYID	CL4	SYSTEM IDENTIFICATION (SMF ID)
18	SMFHSSID	CL4	SUBSYSTEM ID (SDB_ OR SWS_)
22	SMFHSUTY	BL2	RECORD SUBTYPE (05)
24	SMFHVRCD	CL8	SDB/SWS VERSION CODE
32	SMFHRS00	CL8	RESERVED FOR FUTURE USE

Table E–1. SMF Type 05 Fields and Descriptions

*Where xxxxxx represents the version number. For example, `‘NEON.SV040100.CNTL’` is release 4.1 and `‘NEON.SV040500.CNTL’` would be release 4.5.

Offset	Field Name	Field Type or Value	Description
40	SMO5CLIP	CL16	CLIENT IP ADDRESS
56	SMO5SMID	CL4	HOST SYSTEM SMFID
60	SMO5PDSS	CL4	PRODUCT SUBSYSTEM NAME
64	SMO5CLUS	CL8	CLIENT USERID OR BLANKS
72	SMO5AUTH	CL4	CLIENT AUTHORIZATION STATUS • **NONE**. Authorization not sent • **SENT**. Authorization information sent but was not used by the server • **YES**. Client userid/assword were valid • **NO.** Client userid/password were invalid
76	SMO5RS00	CL4	RESERVED FOR FUTURE USE
80	SMO5SRCP	D	CPU TIME USED (TIMEUSED MACRO)
88	SMO5CNID	XL4	CONNECTION ID
92	SMO5LGTM	XL8	TRANSACTION CONNECT TIME (GMT TOD)
100	SMO5ELTM	XL8	TRANSACTION ELAPSED TIME
108	SMO5WRTO	XL8	TOTAL BYTES WRITTEN (RAW)
116	SMO5RS01	XL4	RESERVED FOR FUTURE USE
120	SMO5ADLT	XL8	TRANSACTION CONNECT TIME (LOCAL TOD)
128	SMO5MTCT	F	COUNT OF URL MATCHES PROCESSED
132	SMO5ABCD	XL4	TRANSACTION ABEND CODE (IF ANY)
136	SMO5ABRS	XL4	TRANSACTION ABEND REASON (IF ANY)
140	SMO5TRRC	F	OVERALL RETURN CODE
144	SMO5TRST	F	HTML STATUS CODE
148	SMO5TRRS	F	REASON CODE
152	SMO5IPAD	F	IP ADDRESS OF CLIENT
156	SMO5DBCP	CL8	DB2 CPU TIME
164	SMO5NTCP	CL8	NETWORK CPU TIME
172	SMO5RXCP	CL8	SHADOW/REXX CPU TIME
180	SMO5RPCP	CL8	USER PROGRAM CPU TIME
188	SMO5OHCP	CL8	OTHER CPU TIME
196	SMO5SLCP	CL8	SSL PROCESSING CPU TIME
204	SMO5RS02	CL24	RESERVED FOR FUTURE USE

Table E–1. SMF Type 05 Fields and Descriptions

Offset	Field Name	Field Type or Value	Description
228	SMO5RDTO	XL8	TOTAL BYTES SENT INBOUND
236	SMO5INUR	CL128	ORIGINAL IN-BOUND URL VALUE
364	SMO5RESC	F	COUNT OF URL RESCANS
368	SMO501CR	CL128	WWW RULE CRITERION (URL MATCH STRING)
496	SMO501RS	CL8	WWW RULE EVENT PROCEDURE SET NAME
504	SMO501RL	CL8	WWW RULE EVENT PROCEDURE MEMBER NAME
512	SMO501EU	CL8	RUN-TIME MVS USERID IN EFFECT
520	SMO5LSCR	CL128	WWW RULE CRITERION (URL MATCH STRING)
648	SMO5LSRS	CL8	WWW RULE EVENT PROCEDURE SET NAME
656	SMO5LSRL	CL8	WWW RULE EVENT PROCEDURE MEMBER NAME
664	SMO5LSEU	CL8	RUN-TIME MVS USERID IN EFFECT
672	SMO5USR1	CL256	USER DATA AREA 1
928	SMO5USR2	CL256	USER DATA AREA 2

Table E–1. SMF Type 05 Fields and Descriptions

SMF Type 06 Records

If you have the SMFTRANSACT parameter set to **`YES`** in the SWSxIN00 initialization exec, these records are written for each inbound client request. Each SMF transaction record contains information about all the work done on behalf of the client. The inbound client request may cause zero, one, or more SQL operations to be executed.

A sample SAS program has been provided that can be used to print out these SMF fields. The program is located in the `‘NEON.SVxxxxxx.CNTL’`* dataset, member `SMFSWS06`.

Offset	Field Name	Field Type or Value	Description
	SMFHF	BL1	HEADER FLAG BYTE
	SMFHESA4	B'00010000'	MVS/ESA 4
	SMFHXA	B'00001000'	MVS/XA
	SMFHESA	B'00000100'	MVS/ESA

Table E–2. SMF Type 06 Fields and Descriptions

*Where xxxxxx represents the version number. For example, `‘NEON.SV040100.CNTL’` is release 4.1 and `‘NEON.SV040500.CNTL’` would be release 4.5.

Offset	Field Name	Field Type or Value	Description
	SMFHVS2	B'00000010'	VS2
2	SMFHRCTY	BL1	RECORD TYPE
3	SMFHTIME	BL4	RECORD WRITTEN TIME (TIME BIN)
7	SMFHDATE	PL4	RECORD WRITTEN DATE (0CYYDDDF)
11	SMFHSYID	CL4	SYSTEM IDENTIFICATION (SMF ID)
21	SMFHVRCD	CL8	SDB/SWS VERSION CODE
37	SMO6LNA	CL16	CLIENT SYSTEM NAME
53	SMO6CTY	CL8	CLIENT TYPE (COMMUNICATION TYPE)
61	SMO6IPAD	XL4	CLIENT IP ADDRESS
65	SMO6CLUS	CL8	CLIENT USERID
73	SMO6CNID	XL4	CONNECTION ID
77	SMO6SQOP	XL2	SQL OPERATION CODE
141	SMO6PDSS	CL4	PRODUCT SUBSYSTEM NAME
145	SMO6PLAN	CL8	DB2 PLAN NAME
153	SMO6SSNA	CL4	DB2 SUBSYSTEM NAME
157	SMO6ADLT	XL8	CLIENT LOGON TIME (ADJUSTED FOR GMT)
165	SMO6ADCU	XL8	CURRENT TIME (ADJUSTED FOR GMT)
173	SMO6ELTM	XL8	CLIENT ELAPSED TIME SO FAR (TOD)
181	SMO6SQEL	XL8	CURRENT SQL STATEMENT ELAPSED TIME
189	SMO6SQCP	XL8	CURRENT SQL STATEMENT CPU TIME
197	SMO6SQRC	F	CURRENT SQL STATEMENT RETURN CODE
201	SMO6SQRE	F	CURRENT SQL STATEMENT REASON CODE
205	SMO6SQSQ	F	CURRENT SQL STATEMENT SQL CODE
209	SMO6SQAB	F	CURRENT SQL STATEMENT ABEND CODE
293	SMO6SQLN	F	SQL SOURCE LENGTH
297	SMO6SQSR	F	SQL SOURCE STRING

Table E–2. SMF Type 06 Fields and Descriptions

APPENDIX F: *Language Codes*

The following list of language codes are used to translate text:

Code	Language
BEL	BELGIAN
CBL	CANADIAN BILINGUAL
DAN	DANISH (MS)
DAN2	DANISH/NORWEGIAN
DEU	GERMAN (MS)
DEU2	AUSTRIAN/GERMAN
ENG	U.K. ENGLISH (MS)
ENG2	U.K. ENGLISH
ENU	U.S. ENGLISH
ENU2	U.S. ENGLISH (ORIG. SWS VERSION)
ESN	MODERN SPANISH (MS)
ESP	CASTILIAN SPANISH (MS)
ESP2	SPANISH
FIN	FINISH (MS)
FIN2	FINISH/SWEDISH
FRA	FRENCH (MS)
FRA2	FRENCH
FRC	CANADIAN FRENCH
ISL	ICELANDIC (MS)
ITA	ITALIAN (MS)
ITA2	ITALIAN
JPE	JAPANESE/ENGLISH
NLD	DUTCH (MS)
NLD2	DUTCH
NOR	NORWEGIAN (MS)

Table F–1. Language Codes and Descriptions

Code	Language
PTG	PORTUGUESE (MS)
PTG2	PORTUGUESE
SVE	SWEDISH (MS)
SWF	SWISS/FRENCH
SWG	SWISS/GERMAN

Table F–1. Language Codes and Descriptions

Glossary

The following list is compilation of some of the terms you will find used in NEON's documentation. If you do not find the term you are looking for, the best reference to turn to is the IBM publication: *Dictionary of Computing* (SC20-1699). You may also want to check the glossaries of the manuals listed in "Related Publications."

ACB	**Access Control Block.** A control block that links an application program (for example, a CICS system) to an access method (for example, VSAM or VTAM). In communication with DL/I, an ACB is used only when the underlying access method is VSAM.
ACEE	**Access Control Environment Element.** (CICS for MVS only.) In RACF, a control block containing details of the current user, including user ID, current connect group, user attributes, and group authorities. An ACEE is constructed during user identification and verification.
ACF	**Advanced Communication Function.** A group of IBM licensed programs that uses the concepts of Systems Network Architecture (SNA) including distribution of function and resource sharing.
ADABAS	**Adaptable Database System.** A type of database provided by Software AG.
address space	The range of addresses available to a computer program; the area of virtual storage available to a particular job or started task.
AMODE	**Addressing Mode.** An attribute in MVS and MVS/XA program that refers to the address length that a program is prepared to handle upon entry. In MVS/370, an addresses can be 24 bits in length. In the MVS/XA program, addresses can be 24 bits or 31 bits in length.
APF	**Authorized Program Facility.** A security feature of the MVS operating system that restricts the running of programs that make use of privileged machine instructions.
API	**Application Program Interface.** A set of routines provided in libraries that extends a language's functionality
APPC	**Advanced Program-to-Program Communication.** The general facility characterizing the LU 6.2 architecture and its various implementations in products.
application group name	In IMS/VS, a name that represents a defined group of resources (program specification blocks, transaction names, and logical terminal names).

APPN	**Advanced Peer-to-Peer Networking.** An extension to Systems Network Architecture (SNA). Extends the LU 6.2 peer orientation for end-user services to network control and supports multiple LU types, including LU 2, LU 3, and LU 6.2
ASCH	**Application Scheduler.** MVS application scheduler.
Auto-HTML	See Web Enabling.
Block Connection	A logical connection in which the connection resources (i.e., network session, threads etc.) are held for the lifetime of the connection and in which SQL operations may be grouped. Multiple sends may be issued on a physical network session. Each send is one or more SQL operations (a group). This type of connection is very efficient in network usage (only one connection made and network I/Os are reduced), efficient in CPU utilization (no overhead for multiple connections) but holds mainframe resources (TCBs, threads and TCP/IP sessions) over relatively long periods of time. The number of connections is limited by the number allowed for the scarcest resource.
BMP	**Batch Message Processing.** In IMS/VS, a batch processing program that has access to online databases and message queues.
BMS	**Basic Mapping Support.** Provides most of the input and output facilities required by application programs; allows you to separate the tasks of display design and CICS application programming. BMS interprets generalized application program output commands, and generates data streams for specific output devices. (Such data streams are said to be device dependent.) Conversely, it transforms incoming data streams to a form acceptable to application programs. It obtains information about the format of the data stream for the terminal from the terminal control table terminal entry (the TCTTE) for the task, not from the application program. The same BMS input or output commands in an application program can be used with different kinds of device.
CAF	**Call Attachment Facility.** The component of DB2 used by application programs in any address space to connect the application to DB2.
CCTL	**Coordinator Control Subsystem.** (CICS for MVS only.) In IMS/ESA, the transaction management subsystem that communicates with the DRA, which in turn communicates with DBCTL. In a CICS-DBCTL environment, the CCTL is CICS. The term is used in a number of IMS operator commands that apply to DBCTL, and in the IMS manuals.
CDRM	**Cross Domain Resource Manager.** The functions of the system services control point (SSCP) that control initiation and termination of cross-domain sessions.

CGI	**Common Gateway Interface.** An interface between a client (web browser) and internet connection server that receives input data from standard input, parses the data and translates the escaped characters back into real characters, performs any business process required, and sends a response to the client.
CICS	**Customer Information Control System.** A transaction processing extension to the operating system of IBM mainframe computers that makes it easier to write programs that enter, retrieve, and update data interactively from remote terminal services.
client/server	An application architecture where a remote system (the client) accesses data on a local system (the server).
CMOS	**Complementary Metal-Oxide Semiconductor.** A technology that combines the electrical properties of n-type semiconductors and p-type semiconductors.
COBOL	**Common Business-Oriented Language.** High-level programming language based on English, and used for business applications.
COMMAREA	**Communication Area.** A CICS area that is used to pass data between tasks that communicate with a given terminal. The area can also be used to pass data between programs within a task.
CORBA	**Common Object Request Broker Architecture.** A standard for distributed objects being developed by the Object Management Group. Provides the mechanisms by which objects transparently make requests and receive responses as defined by OMG's ORB. The CORBA ORB is an application framework that provides interoperability between objects built in different languages, running on different machines in heterogeneous distributed environments.
CP	**Control Program.** A computer program designed to schedule and supervise the execution of programs of a computer system.
CPI-C	**Common Programming Interface for Communications.** A type of API interface for LU 6.2.
CPU	**Central Processing Unit.** A processing unit. The part of a computer that includes the circuits controlling the interpretation and execution of instructions.
CS	**Cursor Stability.** An option used with block fetch allowing data changes to take place between the time the data is extracted and the time that it is actually used by the application.
CTDLI	A routine provided by IMS that processes DL/I calls from programs written in the C language.

DASD	**Direct Access Storage Device.** A device in which access time is effectively independent of the location of the data.
DB	**Database.** A collection of data with a given structure for accepting, storing, and providing, on demand, data for multiple users.
DB/DC	**DATABASE/Data Communication.** Type of IMS system that supports database as well as data communication access.
DB2	**DATABASE 2.** An IBM relational database management system. See DBMS.
DBA	**Database Administrator.** The person who maintains the database management system. Database Administration. The act of maintaining a database management system.
DBCTL	**Database Control.** (CICS for MVS only.) An interface between CICS for MVS and IMS/ESA that allows access to IMS DL/I full-function databases and to data entry databases (DEDBs) from one or more CICS systems without the need for data sharing. It also provides release independence, virtual storage constraint relief, operational flexibility, and failure isolation.
DBMS	**Database Management System.** System software for storing, accessing and removing information. A relational DBMS, such as DB2, permits a wide variety of views of the stored information without customer programming.
DBRM	**Database Resource Manager** (for example, DB2, IMS, Oracle, etc.)
DDF	**Distributed Data Facility.** The component of DB2 used to access databases and tables on remote nodes in the network.
ddname	**Data Definition Name.** The name of a data definition statement that corresponds to a data control block containing the same name.
DES	**Data Encryption Standard.** The National Institute of Standards and Technology Data Encryption Standard, adopted by the US Government, allowing only hardware implementations of the data encryption algorithm.
DFP	**Data Facility Products.** A group of IBM supplied access methods and utilities.
DL/I	**Data Language l.** In IMS/VS, the data manipulation language that provides a common high-level interface between a user application and IMS/VS. In VSE and CICS/VS, a database access language.
DMF	**Data Mapping Facility.** A feature of Shadow Direct that allows mapping from various sources. Data maps are created via a series

	of ISPF panels that allow the user to specify a dataset containing a listing of a program that contains a data definition.
DNS	**Domain Name Server.** In TCP/IP, a server program that supplies name-to-address translation by mapping domain names to internet addresses.
DRDS	**Dynamic Reconfiguration Data Set.** In VTAM, a data set used for storing definition data that can be applied to a generated communication controller configuration at the operator's request, or can be used to accomplish dynamic reconfiguration of NCP, local SNA, and packet major nodes.
DSNAME	**Dataset Name.** The term or phrase used to identify the data set.
DSA	**Dynamic Storage Area.** (CICS/VSE only.) System initialization parameter that pre-allocates the CICS dynamic storage area at system initialization.
DSN command	**Data source** (definitions); a DB2-supplied TSO command used to run DB2-based application programs and issue commands to DB2.
DTS	**Dynamic To Static Conversion Facility.** Also known as the Plan-Based Static SQL Conversion Facility. DTS converts dynamic SQL to plan-based static SQL. DTS fully supports plan-based security and is not subject to any restrictions with respect to COMMIT and ROLLBACK (including holding locks across a COMMIT or ROLLBACK).
DTSG	A utility developed by NEON System's UK office that provides an easier to use, graphical front end to the Dynamic to Static Analyzer (DSA) program. DTSG was developed using Visual Basic Version 4.0.
EBCDIC	**Extended Binary-Coded Decimal Interchange Code.** A coded character set of 256 8-bit characters.
ECF	**Enterprise Control Facility.** A management tool that is installed with the Enterprise Server and used to define monitoring and control parameters for the local Enterprise Server or any other Enterprise Server on the network.
EOV	End Of Volume.
ESTAE	**Extended Specify Task Abnormal Exit.** An MVS macroinstruction that provides recovery capability and gives control to the user-specified exit routine for processing, diagnosing an abend, or specifying a retry address.
event	A site-defined action, such as a SQL statement, or CICS, IMS or OS/390-MVS application program.

EXCI	**External CICS Interface.** Used by SHADOW_CICS to connect to the specified CICS region and execute the specified program.
EXEC	A TSO command for running REXX programs; a REXX program.
Fast Logon	A connection startup process where handshaking is kept to a minimum to reduce the number of network I/Os (from 2 to 1). Since assumptions are made about the level of code at each end, code level dependencies exist. If these assumptions are incorrect, the connection will fail.
FTP	**File Transfer Protocol.** A protocol used to request and receive files and file system directory information from another computer.
Group	A sequence of SQL operations that is collected and sent together as one block. A group is terminated by a resultset returning SQL operation (i.e., SELECT or CALL) or a logical unit of work termination (i.e., COMMIT - note: a ROLLBACK will cause the operations to be discarded). Only INSERTs, DELETEs and UPDATEs may be grouped. The maximum grouping allowed is determined at initialization time.
GUI	**Graphical User Interface.** A type of computer interface consisting of a visual metaphor of a real-world scene, often of a desktop.
HFS	**Hierarchical File System.** A disk-based filing system built on a hierarchy of special files called directories or folders. Descends from a main directory, called the root. Each lower level is a subsidiary.
HTML	**Hypertext Markup Language.** a simple markup language used to create hypertext documents that are platform independent. HTML documents are SGML (Standard Generalized Markup Language) documents with generic semantics that are appropriate for representing information from a wide range of domains. HTML markup can represent hypertext news, mail, documentation, and hypermedia; menus of options; database query results; simple structured documents with in-lined graphics; and hypertext views of existing bodies of information.
HTML Extension Facility	A run-time tailoring facility supported by Shadow OS/390 Web Server for text format data files. Provides extremely flexible and easy-to-use support for the creation of customized HTML forms and web pages. You make use of the HTML Extension Facility by including HTML-like statements directly within your source file. When the source file is to be transmitted out-bound, the Shadow OS/390 Web Server evaluates the HTML Extension statements and customizes the information that is actually sent.

HTTP	**Hypertext Transfer Protocol.** Protocol used by the World Wide Web. It allows the retrieval of virtually any digital file, in a format suitable for later rendering the file in its original text, audio, or visual media presentation form.
IDMS	Type of database management system supplied by Computer Associates.
IMS	**Information Management System.** An IBM hierarchical database management system.
Internet	A wide area network connecting many networks to allow the free flow of information between otherwise unconnected and often very incompatible computer systems.
Intranet	A closed subnetwork, based on Internet technology. It operates the same way as the global Internet, but usually exists within the confines of a single organization using private communication pathways. An intranet is used to disseminate information to "authorized" users, such as those within the organization, while preventing some or all access from outside the organization.
IP	**Internet Protocol.** A protocol used to route data from its source to its destination in an Internet environment.
IP Address	**Internet Protocol Address.** A two part address, used by TCP/IP to route information packets from one node in the network to another. Within a TCP/IP network IP addresses must be unique.
I/O	**Input/Output.** Pertaining to input, output, or both; or pertaining to a device, process, or channel involved in data input, data output, or both.
IPCS	**Interactive Problem Control System.** A component of VM (virtual machine) that permits online problem management, interactive problem diagnosis, online debugging for disk-resident CP abend dumps, and problem tracking and reporting.
ISPF	**Interactive System Productivity Facility.** An IBM-licensed program that serves as a full-screen editor and dialogue manager; used for writing application programs, it provides a means of generating standard screen panels and interactive dialogues between the application programmer and terminal user.
ISO	International Standards Organization.
IUCV	**Inter-User Communications Vehicle.** An API used by Shadow Server to communicate with IBM TCP/IP.
JCL	**Job Control Language.** A control language used to identify a job to an operating system and to describe the job’s requirements.

LAN	**Local Area Network.** A computer network located on a user's premises within a limited geographical area.
LPA	**Link Pack Area.** An area of main storage containing re-enterable routines from system libraries. In OS/VS2, an area of virtual storage containing re-enterable routines that are loaded at IPL time and can be used concurrently by all tasks in the system.
LRECL	**Logical Record Length.** In CICS/VS, the length of a logical record, which is a data record sent by one transaction program to another. In VSAM, the length of a unit of information normally pertaining to a single object.
LU	**Logical Unit.** A type of network accessible unit that enables end users to gain access to network resources and communicate with each other.
LU 6.2	**Logical Unit 6.2.** An SNA defined protocol for communication between two applications.
LUOW	**Logical Unit Of Work.** In IMS/VS, the processing unit that a program performs between synchronization points.
LZ	**Lempel Ziv.** A type of compression based on repeated characters in the data.
Message Connection	A logical connection in which the connection resources (i.e., network session, threads etc.) are held for the duration of a LUOW in which SQL operations may be grouped. Only one send may be issued on a physical network session. Each send must be a LUOW. INSERTs, DELETEs and UPDATEs cannot be mixed with SELECTs or CALLs without intervening COMMITs or ROLLBACKs.). This type of connection increases network usage (multiple connections) and CPU utilization (multiple connections) but releases mainframe resources (TCBs, threads and TCP/IP sessions) after relatively short periods of time. This is the most efficient mode as far as holding of mainframe resources is concerned. The network usage is greater than or equal to that for Transblock mode due to sessions being terminated after each send. The number of connections may exceed the number of actual resources.
MFS	**Message Format Services.** In IMS/VS, an editing facility that allows application programs to deal with simple logical messages instead of device-dependent data, thus simplifying the application development process.
MIB	Management Information Block.
MIME	**Multimedia Internet Mail Extension.** A type of Internet file supported by Shadow OS/390 Web Server.

MQ Series	Middleware which focuses on reliable and guaranteed delivery by continually retrying to send the message even if there has been gateway failure or a network outage. It even survives a restart of the queue manager.
MRO	**Multiregion Operation.** Communication between CICS systems in the same processor without the use of SNA network facilities.
MTS	**Multithreaded Server.** A type of transactional and object broker server.
MUNIX	Combination of UNIX and 0S/390-MVS knowledge.
MVS	**Multiple Virtual Storage.** An operating system for IBM System 370 hardware. Each user of the system is provided a "virtual" address space equal in size to the addressing limit of the machine. Also shorthand notation for MVS/XA (MVS/Extended Architecture) and MVS/ESA (MVS/Enterprise Systems Architecture).
NDS	**NEON Data Stream.** An ODBC-optimized protocol, implemented between the driver and the server components. NDS interacts with the network at the transport layer, thus avoiding the overhead inherent in higher-level network APIs. It also enhances performance in a variety of ways, including compressing the data, minimizing the number of client-to-server round trips, and increasing the network buffer size.
NLS	**National Language Support.** The modification or conversion of a US English product to conform to the requirements of another language or country.
NT	Network Terminal.
OC	Open Client. Type of API. Not supported by Shadow Direct.
ODBC	**Open Database Connectivity.** An API created by Microsoft that allows applications to access data in database management systems (DBMS) using SQL as a standard. This interface was designed to allow a single application to access many different database management systems.
OE	Open Edition.
OLTP	Online-Transaction-Processing.
OS/2	**Operating System/2.** An IBM supplied operating system for IBM personal computers; has many features, such as multitasking, similar to those of mainframe operating systems.
PCB	**Program Communication Block.** An IMS control block that describes an application program's interface to and view of an IMS

	database or, additionally for message processing and batch message processing programs, to the source and destinations of messages. PCBs are defined by the user during PSB generation.
PDS	**Page Data Set.** A method of storing several programs, such as REXX programs, as members of a single data set. In System/370 virtual storage systems, a data set in external page storage in which pages are stored.
Permanent Connection	A logical connection in which the connection resources (i.e., network session, threads etc.) are held for the lifetime of the connection and each SQL operation is transmitted and executed separately (i.e., no grouping). Multiple sends may be issued on a physical network session. Each send is an individual SQL operation. This type of connection is efficient in network usage (only one connection made) and CPU utilization (no overhead for multiple connections) but holds mainframe resources (TCBs, threads and TCP/IP sessions) over relatively long periods of time. The number of connections is limited to the number allowed for the scarcest resource.
PGP	**Pretty Good Privacy.** Allows companies to perform Electronic Data Interchange (EDI) over the Internet with privacy, authentication, and convenience; combines the convenience of the Rivest-Shamir-Adleman (RSA) public key cryptosystem with the speed of conventional cryptography, message digests for digital signatures, data compression before encryption, good ergonomic design, and sophisticated key management.
PL/I	**Programming Language One.** A programming language designed for numeric scientific computations, business data processing, systems programming and other applications.
PO	**Partitioned Organized.** Type of dataset organization.
Port	A 16-bit number used along with IP address to uniquely identify an application on a node within a TCP/IP network.
PSB	**Program Specification Block.** The control block that describes databases and logical message destinations used by an application program. A PSB consists of one or more PCBs.
PTF	**Program Temporary Fix.** A temporary solution or by-pass of a problem diagnosed by IBM as resulting from a defect in a current unaltered release of the program.
PU	**Physical Unit.** The component that manages and monitors the resources associated with a node, as requested by an SSCP via an SSCP-PU session. This term applies to type 2.0, type 4 and type 5 nodes only.

QMF	Query Management Facility.
RACF	**Resource Access Control Facility.** An IBM-licensed program that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources.
RC	**Return Code.** A code used to influence the execution of succeeding instructions; a value returned to a program to indicate the results of an operation requested by that program.
RDBMS	**Relational Database Management Systems.** A type of database management system that stores information in tables – rows and columns – and conducts searches by using data in specified columns of one table to find additional data in another table.
RDT	**Resource Definition Table.** In VTAM, a table describing the characteristics of each node available to VTAM, and associating each node with a network address.
REXX	**Restructured Extended Executor.** An interpretive language used to write command lists.
RFC	Request for Comments.
RPC	**Remote Procedure Calls.** Allows a client to execute a program on a server, with the program being remote to the client.
RR	**Repeatable Read.** An option used with block fetch, allowing many more pages to be locked for update, especially if the number of rows normally extracted by the query is small.
RSA	**Rivest-Shamir-Adleman.** A scheme for public key cryptography.
RSP	Remote Stored Procedures.
SAA	**Systems Application Architecture.** A set of guidelines promoted by IBM for standardizing the design of large pieces of software. It includes a set of user interface guidelines called Common User Access (CUA), as well as guidelines for data communications, programming languages, and procedure libraries.
SAF	**System Authorization Facility.** An MVS facility for routing authorization requests to RACF or equivalent system security packages.
SAM	**Shadow Activity Monitor.** Provides a workstation-based tool for viewing and reporting the Shadow Server logs. SAM functions as a standard Shadow Direct ODBC client.

SDF	**Shadow Diagnostic Facility.** An ISPF-based application, allowing the administrator to view summary and detail information related to connectivity and to take actions to correct connectivity problems. All of the diagnostic, monitoring, and control information can be accessed and updated through the SDF.
SEF	**Shadow Event Facility.** A comprehensive and flexible mechanism for controlling the overall Shadow Direct client/server environment; allows each installation to tailor the execution characteristics of Shadow Direct to whatever level of detail (per-user, per-group, by time-of-day, etc.) is required.
SID	Site ID.
SMF	**System Management Facility.** An optional control program feature of OS/VS that provides the means for gathering and recording information used to evaluate system usage.
SNA	**Systems Network Architecture.** A layered scheme for communication between devices and applications in a network. Applies mainly to IBM networks.
SNMP	Simple Network Management Protocol.
SPUFI	**SQL Processor Using File Input.** An interactive component of DB2, used to query and maintain DB2 databases.
SQL	**Structured Query Language.** A non-procedural language for creating, querying, and maintaining relational databases.
SRB	**Service Request Block.** (CICS for MVS only.) An MVS dispatchable unit.
SRM	**System Resources Manager.** A group of programs that controls the use of system resources in order to satisfy the performance objectives of the installation.
SSL	**Security Socket Layers.** Encryption for the highest client/server security standard in practical use today.
Table	A named DB2 object, consisting of a specific number of columns and zero or more unordered rows of data.
TCB	**Task Control Block.** In CICS for MVS, an MVS control block. A TCB is created for each MVS task. Several TCBs are created for CICS management programs. All CICS application programs and all non-reentrant CICS code run under a single quasi-reentrant TCB.
TCP/IP	**Transmission Control Protocol/Internet Protocol.** A protocol specifically designed to facilitate communications between heterogeneous networks.

Thread	An individual unit of work in OS/390-MVS used for authorization, data access, transaction access, monitoring and control.
TIB	Terminal Information Block.
TMP	**Terminal Monitor Program.** In TSO, a program that accepts and interprets commands from the terminal and causes the appropriate command processors to be scheduled and executed.
TNUF	**Table Name Utilization Facility.** A feature of Shadow Direct that allows on-the-fly modification of table names on a user-by-user basis.
TP	**Transaction Program.** A program that processes transactions in an SNA network.
TPL	**Transport Parameter List.** An API used by Shadow Server to communicate with Interlink TCP/IP.
Transaction Connection	A logical connection in which the connection resources (i.e., network session, threads, etc.) are held for the duration of each LUOW and each SQL operation is transmitted and executed separately (i.e., no grouping). Multiple sends may be issued on a physical network session. Each send is an individual SQL operation. The physical network connect is terminated at the end of a LUOW (i.e., COMMIT or ROLLBACK). This type of connection increases network usage (multiple connections) and CPU utilization (multiple connections) but releases mainframe resources (TCBs, threads and TCP/IP sessions) after relatively short periods of time. The number of connections may exceed the number of actual resources.
TransBlock Connection	A logical connection in which the connection resources (i.e., network session, threads etc.) are held for the duration of each LUOW and in which SQL operations may be grouped. Multiple sends may be issued on a physical network session. Each send is one or more SQL operations (a group). This type of connection increases network usage (multiple connections) and CPU utilization (multiple connections) but releases mainframe resources (TCBs, threads and TCP/IP sessions) after relatively short periods of time. The network usage is less than or equal to that for Transaction mode due to grouping of sent data. The number of connections may exceed the number of actual resources.
TSO	**Timesharing Option.** The interactive timesharing component of the MVS operating system that supports timesharing terminals.
TSS	**Time Sharing System.** A programming system that provides users with conversational online access to a computing system with one or more processing units and simultaneously processes batched jobs.

UDP	**User Datagram Protocol.** In TCP/IP, a packet-level protocol built directly on the Internet protocol layer. Used for application-to-application programs between TCP/IP host systems.
UNIX™	An operating system developed by Bell Laboratories that features multiprogramming in a multi-user environment. The UNIX operating system was originally developed for use on minicomputers but has been adapted for mainframes and microcomputers.
URL	Uniform Request Locator.
VCF	**Virtual Connection Facility.** Allows sharing of OS/390-MVS connectivity resources across a larger user population by transparently switching connections between “real” and “virtual” as the application shifts from active to idle, and vice versa.
Virtual Storage	An operating system technique for providing more addressable storage to programs than is actually available on the hardware.
VM	**Virtual Machine.** A virtual data processing system that appears to be at the exclusive disposal of a particular user, but whose functions are accomplished by sharing the resources of a real data processing system.
VSAM	**Virtual Storage Access Method.** A type of data set maintained by TSO's Access Method Services program. VSAM datasets may be accessed sequentially and randomly.
VTAM	**Virtual Telecommunications Access Method.** IBM mainframe software that implements portions of the Systems Network Architecture (see SNA).
Web Enabling	The execution of online IMS transactions and commands converted by Shadow OS/390 Web Server into HTML format.
WLM	**Work Load Manager.** A component of the OS/390 operating system, first introduced in MVS/ESA 5.1. It is a policy driven manager system of resources that is intended to allow a user to define system performance goals in the same terms that would be used in a service level agreement.
WWW	World Wide Web.
Work Station	A powerful microcomputer typically used for scientific and engineering calculations. A workstation typically has more than four megabytes of RAM, more than 100 megabytes of disk capacity, and a screen with graphics resolution of at least 800 by 1000. Examples are the Sun Sparcstation and IBM RS/6000.

Wrap-Around Trace Facility	A Shadow Server tool, designed to record critical events in the life of each individual transaction process. In addition, the tool is designed to record critical internal information which can be used to debug and correct problems within the Server itself. The wrap-around trace consists of a large block of virtual storage, which can optionally be backed by a data-in-virtual linear dataset. This block of virtual storage is sub-divided into a *status area, a configurable number of event blocks, and a series of vector tables.*

Index

Reader's Comment Form

At NEON Systems, Inc. we are always looking for good ideas. If you have a suggestion or comment regarding any of our publications, please complete this form, and mail or fax it to us at the following address. Thank you.

Please complete the following information, or attach your business card here.

Your Name: ______________________________

Phone Number: ______________________________

Your Company: ______________________________

Address: ______________________________

____

Publication Name: ______________________________

Version *and* Edition Numbers (see page ii)**:** ______________________________

Suggestion/Request: ______________________________

____

____

____

____

____

____

Please mail or fax this page to:

NEON Systems, Inc.
14100 SW Freeway, Suite 500
Sugar Land, Texas 77478, U. S. A.

Fax Number: **(281) 242-3880**