



# Caché WebLink Guide

Caché WebLink version: 4.2  
Document Revised: 4 June 1999

Caché, Caché WebLink, and Caché WebLink Developer are trademarks of InterSystems Corporation. This documentation is the property of InterSystems Corporation. All rights reserved.

[Copyright © 1999, InterSystems Corporation.](#)

## Chapters

1: [Introduction](#)

2: [Manipulating Queries](#)

3: [Managing State-Aware Sessions](#)

4: [Web Event Broker](#)

5: [Network Service Daemon](#)

6: [Systems Management](#)

- [Checking System Status](#)
- [Closing Sessions Manually](#)
- [Testing Server Connections](#)
- [Global Configuration](#)
- [Customizing Error Messages](#)
- [Configuring Server Access](#)
- [Viewing the Event Log](#)

## Appendixes

A: [Installation](#)

B: [Express Configuration](#)

C: [Supplementary Information](#)

D: [Apache Web Servers](#)

E: [PDQweb Compatibility](#)

# Introduction

- [Caché WebLink Architecture](#)
  - [Caché WebLink Connection Methods](#)
  - [State-Aware Caché WebLink](#)
  - [Web Event Broker](#)
  - [Caché WebLink Developer](#)
- 

Caché WebLink provides high performance data exchange between Caché databases, World Wide Web servers, and Web browsers.

**Note:** This document will assume that you are running Caché WebLink against a Caché database server. However, because Caché WebLink will run on Open M systems (such as DSM), everything will also apply to those systems, unless otherwise noted.

Caché WebLink operates in either an Internet or Intranet environment, and includes 2-Byte character support.

Caché WebLink is a standard component of Caché 3.1. It is installed automatically as part of the Standard installation option. If you are running Caché you do not need an additional license for Caché WebLink.

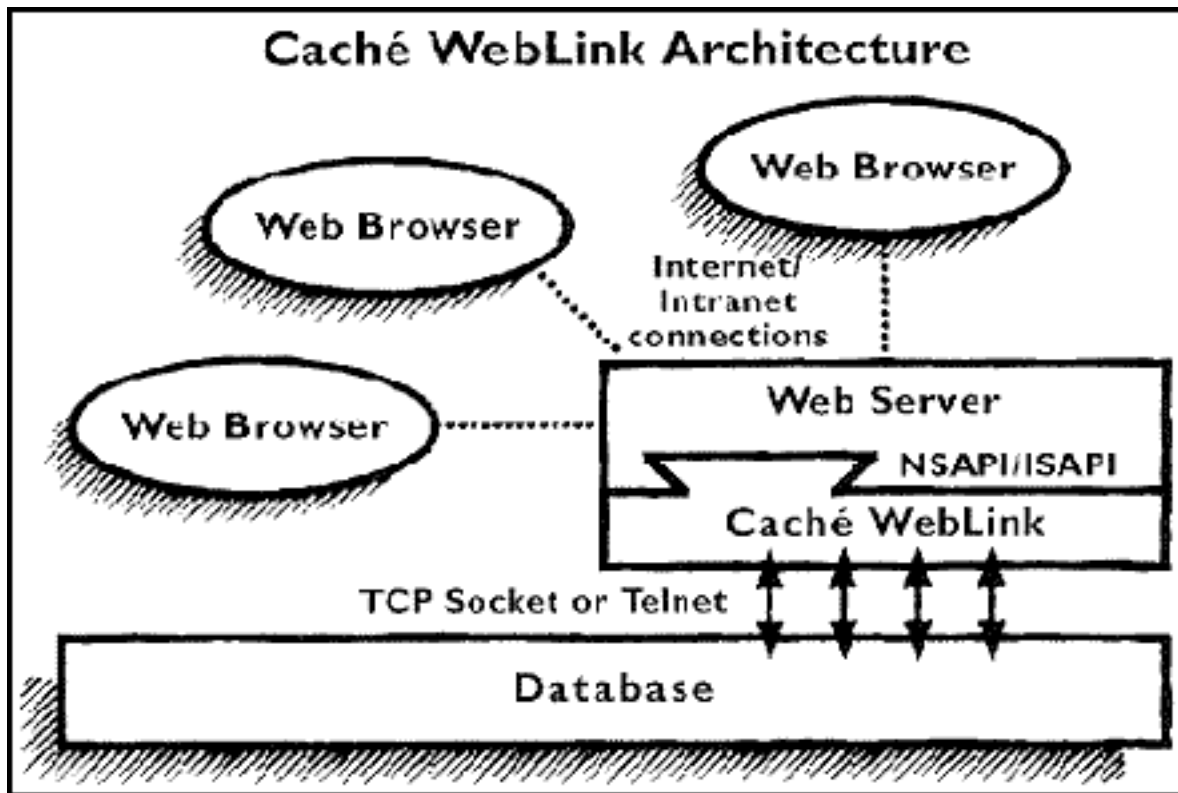
Caché WebLink is also available as an add-on product for ISM, DSM, and MSM systems. A single-connection license for these systems is free of charge.

## Caché WebLink Architecture

Caché WebLink functions in the following manner:

1. Caché WebLink receives a request.
2. Caché WebLink passes the request to a Caché system for processing.
3. Caché WebLink returns the Caché output to the Web client.

The architectural model for this functionality is shown below:



- Dynamic link libraries (DLLs) or shared object libraries manage the interface between the Web server and the universal TCP/IP service for Caché.
- The libraries manage communication between the client and the Caché databases and are responsible for network communications and maintaining state.
- The Caché routine %mgw (the Caché WebLink service daemon, located on the Caché server) interprets and responds to requests received from the client and passes control to and from a selected Caché

- application.
- Caché or Open M systems can be remote or local. You can define up to 32 Caché or Open M systems in your Caché WebLink configuration.

To provide the high performance required by transaction-processing applications, Caché WebLink takes advantage of the two most popular Web server application programming interfaces: NSAPI and ISAPI.

Caché WebLink is also supported for Apache web servers and all web servers capable of running CGI modules.

## Caché WebLink Connection Methods

Caché WebLink provides two connection methods:

- TCP Socket
- TCP/IP TELNET

Each connection type is described in the following two sections.

### TCP Socket Connections

Caché WebLink can establish a connection to a TCP socket, a logical port, by means of a Process Control Daemon (PCD) running within Caché. Caché WebLink socket connections provide a number of significant advantages:

- They can accept binary data, including graphics exported from Caché.
- They provide a much faster connection than TELNET connections.
- They are more secure. When using socket connections, you do not need to disclose a password to your user or to store passwords in your Caché WebLink configuration file.
- They can access data stored on Caché for Windows systems.

The Process Control Daemon can be started automatically from the Caché system startup routine (ZSTU on Caché/ISM systems). For information on starting the PCD, see [Starting the Process Control Daemon](#).

## TCP/IP TELNET Connections

Caché WebLink also establishes connections to Caché systems by using the TCP/IP TELNET protocol. Telnet connections do not require a separate process daemon running on Caché to establish connections to the Caché system.

## State-Aware Caché WebLink

In traditional Web client/server interaction, the Web server receives a request, responds to the request and "forgets" the request. When the client makes additional requests, the Web server has no knowledge of any previous requests the client has made. This is a **stateless** connection.

Caché WebLink features a **state-aware** option that allows the user to interact with a Caché system via the Web as if the user were using a terminal connected to the Caché system. When a Web client connects to a Caché system, the client can choose to use state-aware connection which binds the session to a specific socket or telnet connection allowing normal Caché state-aware transactions. For information on programming for state-aware sessions, see [Managing State-Aware Sessions](#).

**Note1:** On UNIX systems, you can not start multiple httpd processes to serve state-aware connections. You must use multiple threads for your server, but you should use only one Netscape process. For information on setting your Web server to use a single process, see [Preparing Your Systems for Caché WebLink Installation](#).

**Note2:** On UNIX systems, do not specify a value for "process life-span" in your Netscape configuration. This parameter defines the number of requests that each Web process will serve before shutting down and can adversely affect state-aware session. This is defined in the **magnus.conf** file. By default it is not set.

## Web Event Broker

Caché WebLink includes a customizable Java applet that enables real-time "per-field" input verification and substitution. The part of WebLink that processes communications to and from the applet is called the **Event Broker**.

With the applet and the Event Broker, applications will be even more interactive, because users can get instant feedback about input as they are entering it, without waiting to submit an entire page.

## Caché WebLink Developer

Caché WebLink includes Caché WebLink Developer, a component that allows rapid development of dynamic Web applications from HTML files. Using off-the-shelf Web page creation tools (such as Microsoft's FrontPage), developers can design an application as a set of Web pages with embedded Caché ObjectScript code.

Caché WebLink Developer compiles these pages into a Caché-based Web application. At runtime, variables defined in the application are replaced with appropriate data from the Caché database.

For details, see the *Caché WebLink Developer Guide*, which is a separate document shipped with the Caché 3.1 online documentation set.



# Express Configuration

This section contains directions for *quickly* configuring your Caché WebLink setup. The advanced configuration features of the [Systems Management utility](#) are intended for Caché WebLink global configuration and tuning.

Express configuration is divided into three sections, based on each type of connection:

- [Express Configuration for Socket Connections](#)
- [Express Configuration for TELNET Connections](#)
- [Express Configuration for TCP Sockets with TELNET Startup Connection](#)

For information on which type of connection is best for you, read [Caché WebLink Connection Methods](#).

This section also describes how to [start up the PCD](#).

## Express Configuration For Socket Connections

This section explains how to add entries to your Caché WebLink configuration for each server configuration using TCP sockets.

1. From the Caché WebLink Systems Management page, select **Configure Server Access**.
2. In the **Server Name** textbox, enter a unique, descriptive name for the server. This logical name is used to identify each server configuration in the Caché WebLink configuration file.

**Note:** Do not use **SYSTEM** or **SYSTEM\_INDEX** as server names, because these names are reserved.

3. Click the **Configure Server** button. The second configuration screen appears.
4. For the Connection\_Method parameter, select the **TCP Sockets** radio button.
5. Select the Server\_Type (for example: InterSystems Caché 3.x).
6. Enter the IP address of the Caché server. Caché WebLink does not accept the fully-qualified domain name (FQDN) or any aliases.
7. In the **TCP Socket Parameters** section, fill out these options:
  - For **Base\_TCP\_Port**, specify the listening port of the PCD. InterSystems recommends the default port of 1972.
  - For **PCD\_NameSpace**, specify the namespace of the Process Control Daemon. This will generally be the system manager's UCI or namespace.
  - For **Default\_User\_NameSpace**, specify the namespace or UCI that the users using this database have. Note that a UCI on a DSM system corresponds to a namespace on an Caché system.
  - For **NLS\_Translation**, select the translation table your database uses if your system uses something other than Latin1 (**Caché 3.x systems only**).
8. When you have finished, click the **Save This Configuration** button.

Caché WebLink is now installed and configured. InterSystems recommends that you [test the database connectivity](#).

## Express Configuration for TELNET Connections

This section explains how to add entries to your Caché WebLink configuration for each server configuration using TELNET.

1. From the Caché WebLink Systems Management page, select **Configure Server Access**.
2. In the **Server Name** textbox, enter a unique, descriptive name for the server. This logical name is used to identify each server configuration in the Caché WebLink configuration file.  
  
**Note:** Do not use **SYSTEM** or **SYSTEM\_INDEX** as server names, because these names are reserved.
3. Click the **Configure Server** button. The second configuration screen appears.
4. For the Connection\_Method parameter, select the **Telnet** radio button.
5. Select the Server\_Type (for example: InterSystems Caché 3.x).
6. Enter the IP address of the Caché server. Caché WebLink does not accept the fully-qualified domain name (FQDN) or any aliases.
7. For the Login\_Profile parameter, enter a username, password, and server prompt combination. You can either enter it directly as described in the [Configuring Server Access](#) chapter or click the **Generate** button.

If you select **Generate**, Caché WebLink initiates a telnet session to the target server. This session is visible in the box at the top of Login\_Profile Generation screen. Do not enter information into this box. Below this box is an input box. Type the login information into this box as you are prompted for it. Log in with the same login name and password that you want the Caché WebLink client to use. When you see the Caché or Open M prompt, disconnect from the server using the **Press on Entering M** button. Caché WebLink stores the information in the Login\_Profile section of the **mgw.ini** file.

**Caution:** Do not use the **Back** navigation button because this may interfere with filing the configuration.

8. Click the **Save This Configuration** button.

Caché WebLink is now installed and configured. InterSystems recommends that you [test the database connectivity](#).

## Express Configuration for TCP Sockets with TELNET Startup Connections

This section explains how to add entries to your Caché WebLink configuration for each server configuration using TCP Sockets with a TELNET connection starting the PCD.

1. From the Caché WebLink Systems Management page, select **Configure Server Access**.
2. In the **Server Name** textbox, enter a unique, descriptive name for the server. This logical name is used to identify each server configuration in the Caché WebLink configuration file.  
  
**Note:** Do not use **SYSTEM** or **SYSTEM\_INDEX** as server names, because these names are reserved.
3. Click the **Configure Server** button. The second configuration screen appears.
4. For the Connection\_Method parameter, select the **TCP Sockets with Telnet Startup** radio button.
5. Select the Server\_Type (for example: InterSystems Caché 3.x).

6. Enter the IP address of the Caché server. Caché WebLink does not accept the fully-qualified domain name (FQDN) or any aliases.
7. In the **TCP Socket Parameters** section, fill out the options, as [described above](#).
8. Fill in the the Login\_Profile parameter, as [described above](#).
9. Click the **Save This Configuration** button.

Caché WebLink is now installed and configured. InterSystems recommends that you [test the database connectivity](#).

## Starting the Process Control Daemon

**Note for users of Caché version 3.x:** If the Caché super-server is operational on your system **and** you have defined your PCD to listen on TCP port number 1972, the super-server will automatically start WebLink's PCD for you when the first client request comes in for Weblink. Therefore, no action is required here.

The following command shows the general form for starting the PCD. Modify the command appropriately for your system and place it in your startup routine (ZSTU on Caché systems) or type it at the command line.

```
J pcd^%mgw( "Database_Type" ,TCP_Port ) [ "UCI" ]
```

For example, on a Caché system using the default PCD port and a namespace of user, the command would look like this:

```
J pcd^%mgw( "ISC" ,1972 ) [ "user" ]
```

If you do not specify a namespace or UCI, the PCD will start in the manager's namespace or UCI.

When the Web server shuts down, WebLink will close all PCDs that were started by WebLink. Any PCDs started by the Caché server will remain active.



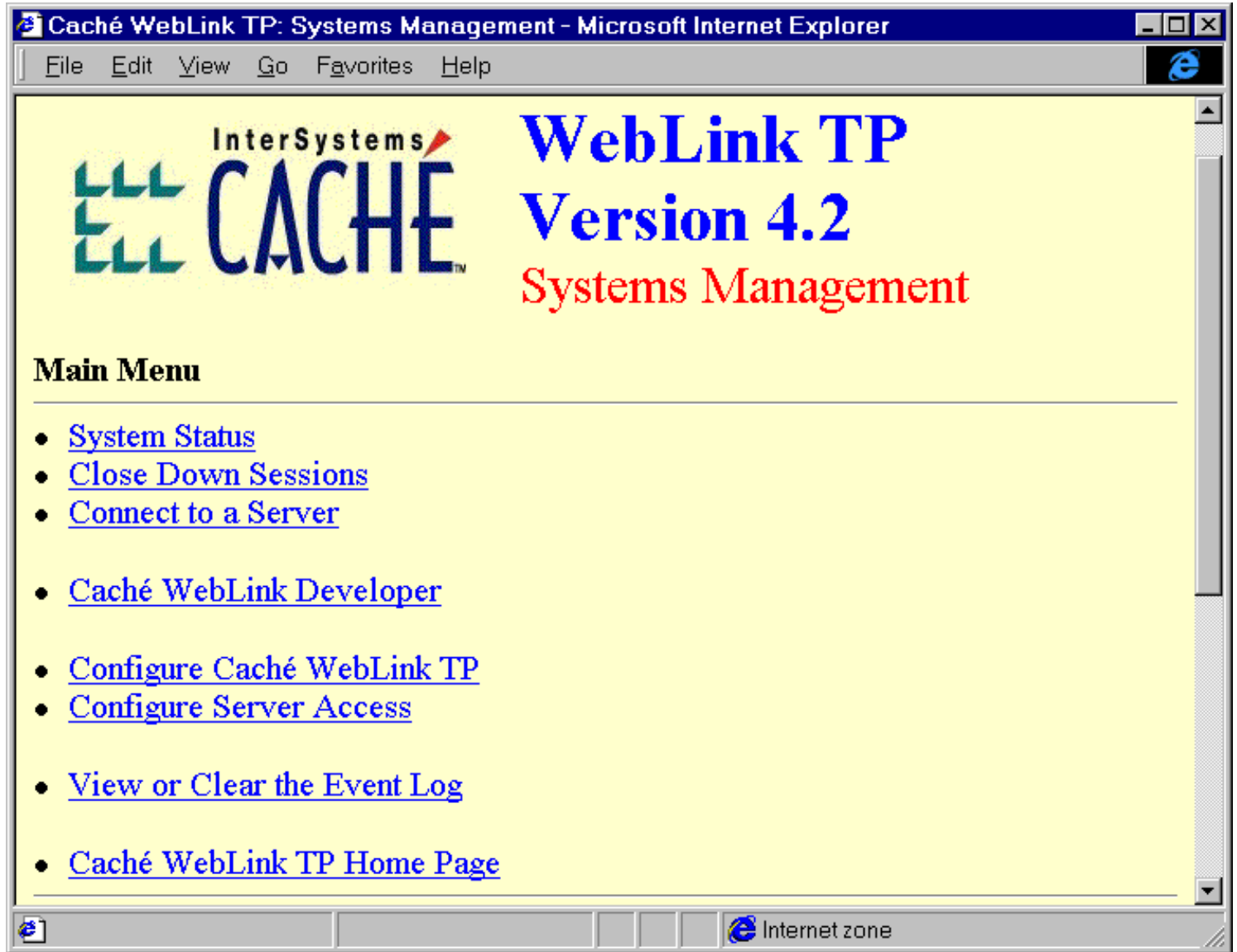
## Caché WebLink Systems Management

You configure and manage Caché WebLink from the Systems Management Main Menu. You can access this screen using a URL composed of IP number or hostname and the directory path where the Caché WebLink software resides on the Web server.

For example, using a Web client on the same system as your Caché WebLink Web server extensions, you could use a URL similar to this:

```
<A HREF="http://127.0.0.1/scripts/mgwms32.dll">Caché WebLink Systems Management</A>
```

The Systems Management Main Menu looks like the following example:



The following table lists the options on the main menu.

Option	Meaning
<a href="#">System Status</a>	Displays the status of active Caché WebLink server processes and sessions.
<a href="#">Close Down Sessions</a>	Closes all sessions on all or specified Caché or Open M servers.
<a href="#">Connect to a Server</a>	Opens a state-less or state-aware session to a Caché or Open M server.
<a href="#">Caché WebLink Developer</a>	Lets you run aché WebLink Developer on a specific Caché or Open M server. For details, see the <i>Caché WebLink Developer Guide</i> .
<a href="#">Configure Caché WebLink TP</a>	Lets you configure Caché WebLink on a specific Caché or Open M server. Also lets you customize Caché WebLink responses to errors and other conditions.



<a href="#">Configure Server Access</a>	Configures Caché WebLink's access to a specific Caché or Open M server. Also lets you disable access to a server.
<a href="#">View or Clean the Event Log</a>	Lets you view information in the Caché WebLink log, as well as clear its contents.
<a href="#">Caché WebLink TP Home Page</a>	Connects to the Caché WebLink home page on the InterSystems Web site.

The following sections describe these options.



## Checking System Status

The **System Status** option on the Systems Management Main Menu displays the status of any active Caché WebLink sessions. You must be a system manager to use this feature.

The following is an example of a System Status output:

Session Number	Server Name	IP Address	TCP Port	Server Type	Server NameSpace	Server Process ID	Status	Activity	Close
0	LOCAL	127.0.0.1	1973	Caché 3.x	USER	253	Free	1	Close
1024	*PCD* LOCAL	127.0.0.1	1972	Caché 3.x	%SYS	244	InUse	1	Close
1056	Web Event Broker		7001				Free	0	
Refresh									

The Status screen displays the following information:

**Session Number**

The number that Caché WebLink assigns to each session at session startup. Your Caché WebLink license determines the number of possible sessions.

**Server Name**

The name of the running Caché system configuration. This column also indicates if the connection is a **PCD** (Process Control Daemon) or **Web Event Broker**.

**IP Address**

The IP address of each Caché system configuration.

**TCP Port**

The TCP port on which each Caché WebLink session connects. The default TELNET port is 23. The default socket port is 1972 for the first connection to a Caché system.

**Server Type**

The type of InterSystems system product running. For example, it reads **Caché 3.x** for a Caché 3.1 system.

**Server Namespace**

The Caché namespace (such as %SYS or USER), DSM UCI, or directory tied to the Caché WebLink process.

**Server Process ID**

The process ID that Caché assigned to the query.

**Status**

Indicates if information is being sent to or from the Caché system. If this column reads **Free**, no information is being sent. If it reads **In Use**, the session is transmitting information. If it reads **Private**, the session is state-aware and not free for general use.

**Activity**

The number of transactions the session has processed.

**Close**

The **Close** button (if available) lets you close down the session.

**Refresh**

The **Refresh** button lets you update the screen.

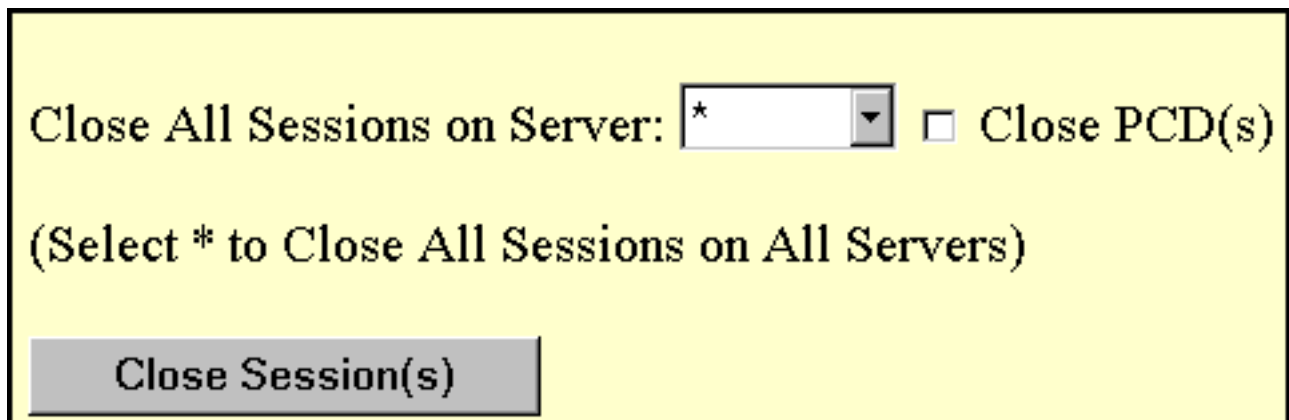
## Closing Sessions Manually

If the **Session\_Reuse** parameter is set to FALSE, the Caché WebLink session closes automatically after Caché WebLink receives a response from the Caché system; otherwise, the system remains active waiting to serve the next request from a browser. However, under certain circumstances, you may want (or need) to close down Caché sessions manually.

If your Caché system shuts down while a Caché WebLink session is still active, Caché WebLink will continue to try to connect to the system until one of the following occurs:

- It successfully reconnects to the system
- Caché WebLink is shut down
- The connection is manually closed

If your Caché system is scheduled for extensive down-time, you may want to close the session from the **Close Down Sessions** menu:



Close All Sessions on Server: \* ☐ Close PCD(s)

(Select \* to Close All Sessions on All Servers)

Close Session(s)

Note that you can close the session while the Caché system is down.

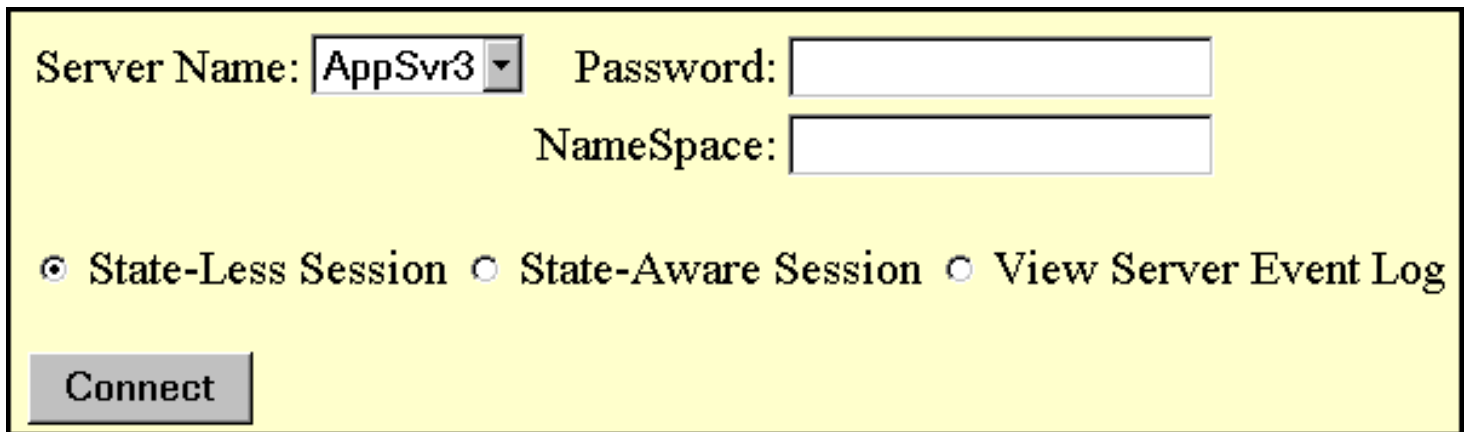
To close sessions manually:

1. From the Systems Management Main Menu, select **Close Down Sessions**.
2. Use the drop-down listbox to select the server configuration from which to disconnect. Select \* (asterisk) to close all sessions to on all Servers.
3. Check the **Close PCD(s)** box to close down the Process Control Daemon for any configuration or all configurations.
4. Click the **Close Sessions(s)** button.

Note that only people with system manager access can use this feature.

## Testing Server Connections

The **Connect to a Server** option on the Systems Management Main Menu is useful to test Caché WebLink connectivity to your systems. The screen looks like the following example.



Server Name: AppSvr3 Password:

Namespace:

☒ State-Less Session ☐ State-Aware Session ☐ View Server Event Log

Connect

Note that you must be a system manager to use this feature.

To test Caché WebLink connectivity:

1. Open the Caché WebLink Systems Management page of the Web server on which Caché WebLink is running.
2. Select **Connect to a Server**.
3. Select the desired configuration from the displayed list.
4. Enter a password for the server (if needed).
5. Enter the namespace to connect to (optional).
6. Specify one of the following:
  - **State-Less Session** for the connection.
  - **State-Aware Session** for the connection.
  - **View Server Event Log** to just view the event log of that server.
7. Click the **Connect** button.

Depending on your selection and the state of the server connection, you will receive one of the results listed below.

### Connectivity Results

The **Connect to a Server** option returns one of the following responses.

#### Caché WebLink Test Page

Caché WebLink is installed and working properly. You should receive this page if you selected **State-Less Session** or **State-Aware Session**. The page lists the variables (both user-defined and CGI Environment) that were passed to the server.

#### Server Event Log Page

You should receive this page if you selected **View Server Event Log**.

#### System Availability Error.

This error occurs any time that Caché is unreachable. If there are no additional error messages, check to ensure your Caché system is running.

**Server is currently unavailable**

Caché WebLink may have timed out when trying to establish a new telnet connection. Try increasing the [Login\\_Response\\_Timeout](#) configuration parameter.

**No response from the Web Server****File Empty**

Check to see if your Web server is running. If it is running, shut it down and restart it.

**Not Enough Sessions****All channels to the server are busy: Please try later**

You have reached your session limit and must [close](#) some of the sessions you have running.

If you have a single-connection evaluation license, you might receive this error frequently, unless you set the value of the [Session\\_Reuse](#) configuration parameter to FALSE.

**A Dialog box with this text: Unknown File Type - You have started to download a file of type application/x-unknown-content-type-dllfile....**

You are running a Netscape browser with a Microsoft Web server. You probably need to customize the Web server to prevent this error.

Alternatively, your Web server expects an HTTP header to specify the type of information in the file. (An example of an HTTP header for a Caché application appears elsewhere in this document.) Caché WebLink provides an option to insert a minimal HTTP header into your Web pages if the HTTP header is not provided. If you receive this error message frequently, configure this option.

If these solutions do not solve the problem, [send e-mail](#) to InterSystems Technical Support (support@intersys.com).



## Global Configuration

The **Configure Caché WebLink TP** option on the Systems Management Main Menu provides you with a mechanism for a global (system-wide) configuration of Caché WebLink. This option is mainly intended for performance tuning, and therefore should be used only by advanced users. Note that you must be a system manager to use this option.

The set of parameters that you establish with this option is the **global configuration** for Caché WebLink. This means that when you configure server access parameters, any unspecified optional parameters and/or custom system forms are automatically inherited from the global configuration. For example, if you do not set a `Server_Response_Timeout` parameter for a specific server, that server will inherit the global `Server_Response_Timeout` setting.

Caché WebLink advanced configuration consists of two components:

- configuring system parameters (see below)
- [customizing error messages](#)

The top half of the configuration screen allows you to specify system parameters (only part of the screen is shown below).

Caché WebLink TP: Systems Management - Microsoft Internet Explorer

File Edit View Go Favorites Help

InterSystems  
**CACHE**

**WebLink TP Version 4.2**  
**Systems Management**

**Configure Caché WebLink TP** [Return to Main Menu](#)

Site\_Name: WebApp

Web\_Server\_Ip\_Address: 127.0.0.1

Max\_Sessions: 1

Max\_Machines: 1

Login\_Response\_Timeout: 30

Server\_Response\_Timeout: 30

Queued\_Request\_Timeout: 30

No\_Activity\_Timeout:

Default\_Service: LOCAL

Internet zone

To set advanced configuration parameters:

1. From the Systems Management Main Menu, select **Configure Caché WebLink TP**.
2. Set one or more system parameters.
3. Click the **Save Configuration** button.

The Caché WebLink system parameters are:

### Site\_Name

The name of the site as described at purchase time or during installation. This name appears in the **Caché WebLink Licensed to** section of all HTML pages generated internally by the Caché WebLink libraries. It must match the details provided when the license was purchased. The `Site_Name` field is case sensitive. If the site name does not match the name of the site the license key was issued for, Caché WebLink defaults to a single-user license.

#### Web\_Server\_Ip\_Address

The IP address of the Web server. It must be the physical IP Address of the Web server and is checked against the license. If the IP addresses do not match, Caché WebLink defaults to single-connection evaluation status regardless of how many connections you purchased. Because of this checking, licenses are non-transferable. Contact InterSystems if you need to change licenses to accommodate a change in IP Address.

#### Max\_Sessions

The maximum number of concurrent sessions allowed to connect to Caché WebLink configurations. The maximum number of allowable connections is determined by the license and must match the number given when you purchased the license.

#### Max\_Machines

The maximum allowable number of individual Caché WebLink systems. An individual Caché system is identified by its IP\_Address. This means that multiple configurations for a single system are considered as one machine.  
For example, one group of users may need access to billing information while another group needs personal records. This information is located in the same Caché system. You can set up two Caché WebLink configurations reflecting the needs of the groups. Because both configurations point to the same IP address, this counts as a single machine in the Max\_Machine field.  
The maximum allowable value for this parameter is 32.

#### Login\_Response\_Timeout

The maximum number of seconds allowed for the target server to respond to each prompt in the login sequence, as defined in the server's [Login\\_Profile](#) parameter. The minimum allowable value is 5 seconds.  
The value set here is the default for the system. However, you may set a different value on individual server-specific configurations.

InterSystems recommends that you do not merely set the timeout to a high number. Caché WebLink uses the Login\_Response\_Timeout to decide if Caché is responding. Caché WebLink always makes three attempts to connect. If the connection attempt fails, Caché WebLink sends an error message.

#### Server\_Response\_Timeout

The maximum number of seconds allowed for the target server to respond to a request from the Web server. The timeout refers to a period of no activity, so, for example, sending a line of HTML data every second for 10 hours does not cause a timeout. The minimum allowable value for this field is 5 seconds.

The value set here is the default for the system. However, you may set a different value on individual server-specific configurations. You can also override the global configuration setting by using the reserved key word **MGWSTO** within your applications. For more information, see [Reserved Key Names in HTML forms](#).

Note that if a target server is restarted, Caché WebLink tries to reconnect the session automatically. If a response does time out, Caché WebLink assumes an error, terminates the process and returns an error page to the user. If your request requires a significant amount of Caché processing, increase the value of this parameter accordingly.

#### Queued\_Request\_Timeout

The maximum number of seconds that a request can remain in a process queue waiting for an available connection to the appropriate Caché system. The minimum allowable value is 5 seconds.

If a request for a Caché system is queued and subsequently times out, Caché WebLink sends an error message ("Server busy - try again later") to the client.

#### No\_Activity\_Timeout

This parameter is relevant to **state-less** connections only.

This parameter indicates the maximum amount of time (in seconds) that a state-less session will remain open in an idle state. If this time-out is exceeded, the session will automatically close down. This facility prevents state-less sessions accumulating on your Caché server - particularly after periods of high activity where a large number of sessions may have been opened to cope with the increased work-load. If a value is not specified, state-less sessions will remain open until they are manually closed.

#### Default\_Service

The logical name of a participating Caché WebLink configuration that a client connects to if the client does not specify a server.

#### Env\_Parameters

The environment control parameters: a string of characters, each representing a command. These parameters are used mainly for troubleshooting. Caché WebLink writes the output of the following command set to a serial log file called **mgw.log**. You can also view or clear the log with the [log viewer](#). Each entry in the log file is time stamped.

InterSystems recommends not enabling environment parameter error logging unless directed otherwise by InterSystems technical support.

The value set here is the default for the system, however you may set a different value on individual Caché WebLink server configurations. For more information see Env\_Parameters in [Configuring Server Access](#).

##### Commands:

##### **L** or **l**

Record the results of the license validation and the system prompt of each Caché system as a new connection is made.

##### **E** or **e**

Record all errors. Caché WebLink is designed to be error tolerant; however, this option allows you to monitor connection failures.

##### **V** or **v**

Verbose: Record entire login dialog between Caché WebLink and a Caché system. This includes passwords, so be sure that your log file is not world readable on multiuser systems.

##### **T** or **t**

Record telnet option negotiation that takes place between Caché WebLink and the remote system. The negotiation is usually communicated in hexadecimal, so the log is written in this form.

If you entered all possible commands into the Env\_Parameters entry, it might look like this:

```
e l v t
```

#### System\_Manager

Specify a system manager's machine for this Caché WebLink configuration, which lets you restrict access to the Caché WebLink configuration files and grant remote configuration access. Enter the IP address of any client machine with configuration access. You can allow system manager access from any number of client machines; however, any client with system manager access can add or delete access to any Caché WebLink system, change any setting in the configuration file, and close down any active sessions. For each additional client, use the form **+IPnumber**. In the example below, if you inserted this line into the system configuration form in the System\_Manager blank, two clients would have system manager access.

```
127.0.0.1+127.1.2.4
```

If this field is undefined, only a client on the same machine as the CachéWebLink Web server extension can configure CachéWebLink.

#### Web\_Event\_Broker\_Port

The port number on which the Web Event Broker will run.

#### Number\_Of\_Event\_Brokers

The number of Web Event Brokers that will run on this configuration. A maximum of eight Event Brokers are supported.

#### **Session\_Reuse**

Allows you to keep Caché Web sessions open to receive requests during a Web session. This option is ignored for state-aware connections.

This value is TRUE (the default) or FALSE.

If set to TRUE, Caché WebLink does not close a session unless explicitly instructed to. The session remains open ready to accept the next request from the Web server. This facility significantly improves the performance of subsequent Web server requests because it eliminates the overhead of establishing a session. The number of sessions depends on the workload prescribed by the Web server. All sessions terminate automatically if the Web server shuts down.

If set to FALSE, then for each Web server request, Caché WebLink opens a new session to service the request and then automatically close the session down when the request has been answered.

The recommended value for this field is TRUE unless you are using a single-connection evaluation license and require access to multiple Caché WebLink systems. The value set here is the default for the system, however you may set a different value on individual server-specific configurations.

#### **Insert\_HTTP\_Headers**

Allows WebLink to insert a minimal HTTP header in all documents not already containing an HTTP header.

This value is TRUE (the default) or FALSE. It is recommended for best portability of applications that you set this to TRUE.

#### **License**

Allows you to review or enter your Caché WebLink license key. If you do not have a license key, this field is blank.





## Customizing Error Messages

The **Configure Caché WebLink TP** option on the Systems Management Main Menu lets you customize the Caché WebLink error messages and system responses. These can be set on a global or per-configuration basis. Note that you must be a system manager to use this option.

The **Custom System Forms** section of the configuration screen is where you can change the default responses (only part of the screen is shown).

**Custom System Forms:**

Error\_No\_Memory:  *Not enough memory to service the request*

Error\_All\_Channels\_Busy:  *All channels to the Server are busy: Please try later*

Error\_System\_Not\_Defined:  *Server cannot be identified*

Error\_Not\_State\_Aware:  *WebLink is not licensed to use State-Aware (TP) connections on this Server*

Error\_Not\_Sockets:  *WebLink is not licensed to use more than one TCP Socket connection on this Server*

Error\_System\_Disabled:  *Server is currently disabled*

Error\_System\_Unavailable:  *Server is currently unavailable*

To customize the default Caché WebLink responses:

1. From the Systems Management Main Menu, select **Configure Caché WebLink TP**.
2. In the **Custom System Forms** section, enter the name of the Web page that contains the new message.
3. Click the **Save Configuration** button.

The Caché WebLink system responses that you can customize are:

### **Error\_No\_Memory**

This message displays when the Web server does not have enough memory to service the request.

### **Error\_All\_Channels\_Busy**

This message displays when all available WebLink connections are in use.

### **Error\_System\_Not\_Defined**

This message displays when WebLink receives a request for an undefined or unidentifiable server.

### **Error\_Not\_State\_Aware**

This message displays when a State-Less WebLink (not licensed to receive state-aware connections) receives a request for a State-Aware session.

**Error\_Not\_Sockets**

This message displays when additional TCP socket connections are attempted using a WebLink licensed for a single-socket connection.

**Error\_System\_Disabled**

This message displays when WebLink receives a connection request for a server marked as disabled.

**Error\_System\_Unavailable**

This message displays when WebLink does not get a reply from the target server.

**Error\_Illegal\_SA\_Access**

This message displays when a client tries to connect to a State-Aware session owned by a different client.

**Error\_Incorrect\_SA\_Handle:**

This message displays when when a client sends WebLink an incorrect State-Aware session identifier.

**Error\_SA\_Session\_Closed**

This message displays when a client tries to connect to a closed State-Aware session.

**Error\_User\_Program**

This message displays when the Web server returns an error. The system manager can discover the cause of an error by checking the event log, if logging is enabled.

**Error\_No\_Data\_Returned**

This message displays when WebLink receives no data from the Web server within the specified time-out period.

**Information\_SA\_Logout**

This message displays when you log out of a State-Aware session.

**Error\_No\_SM\_Access**

This message displays when an unauthorized client attempts to access the Caché WebLink Systems Management pages.

**Information\_Sessions\_Closed**

This message displays when WebLink closes open sessions in response to a client request.



## Configuring Server Access

The **Configure Server Access** option lets you:

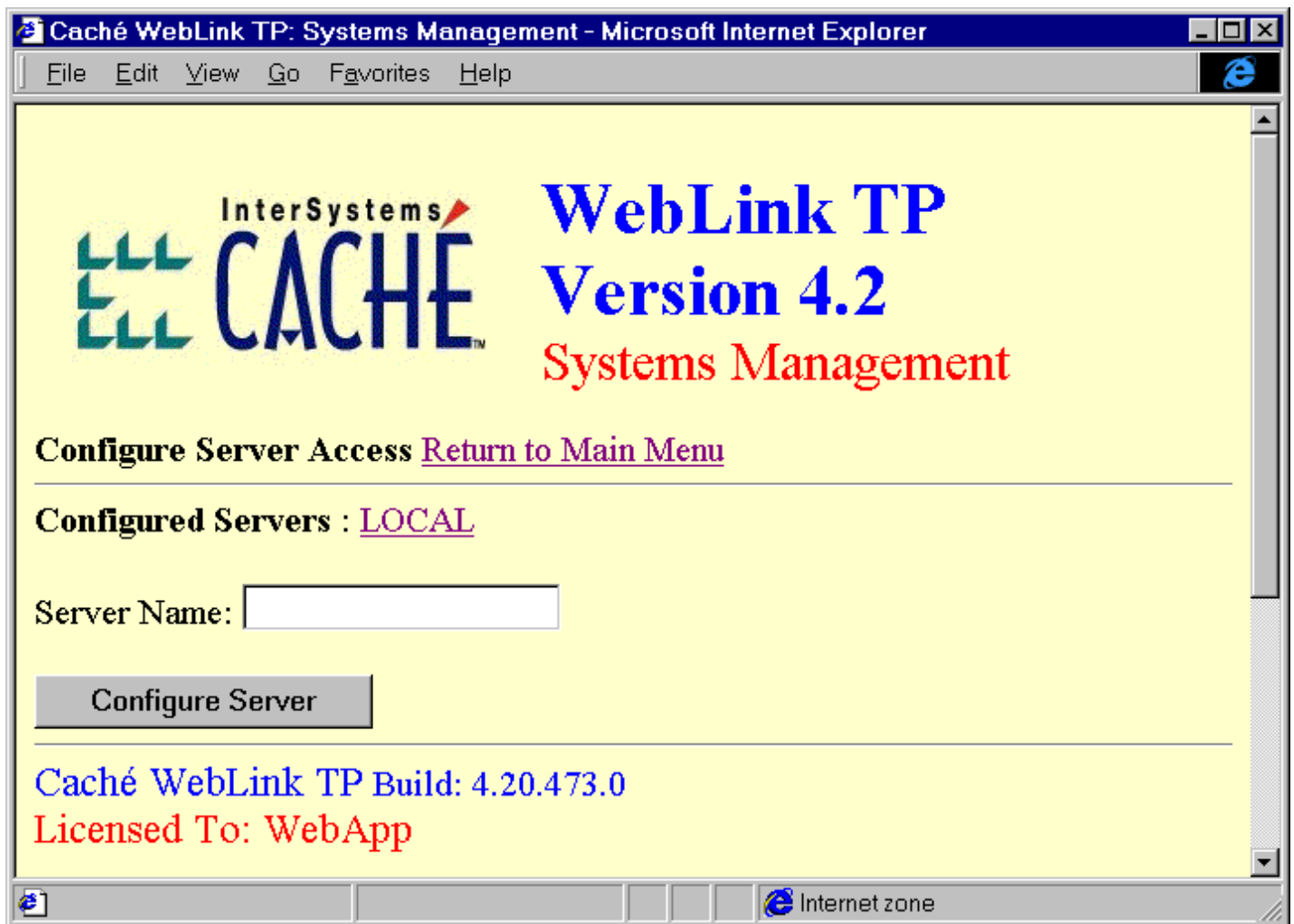
- [Add access](#) to Caché servers to this Caché WebLink configuration, so that your users can access those servers using any Web browser. This method assumes that you will be specifying individual system parameters.
- [Copy](#) a configured server's entry to another name. This is a quick method of adding a new server.
- [Disable access](#) to a configured Caché server.
- [Delete](#) a configured server entry.

### Adding Servers

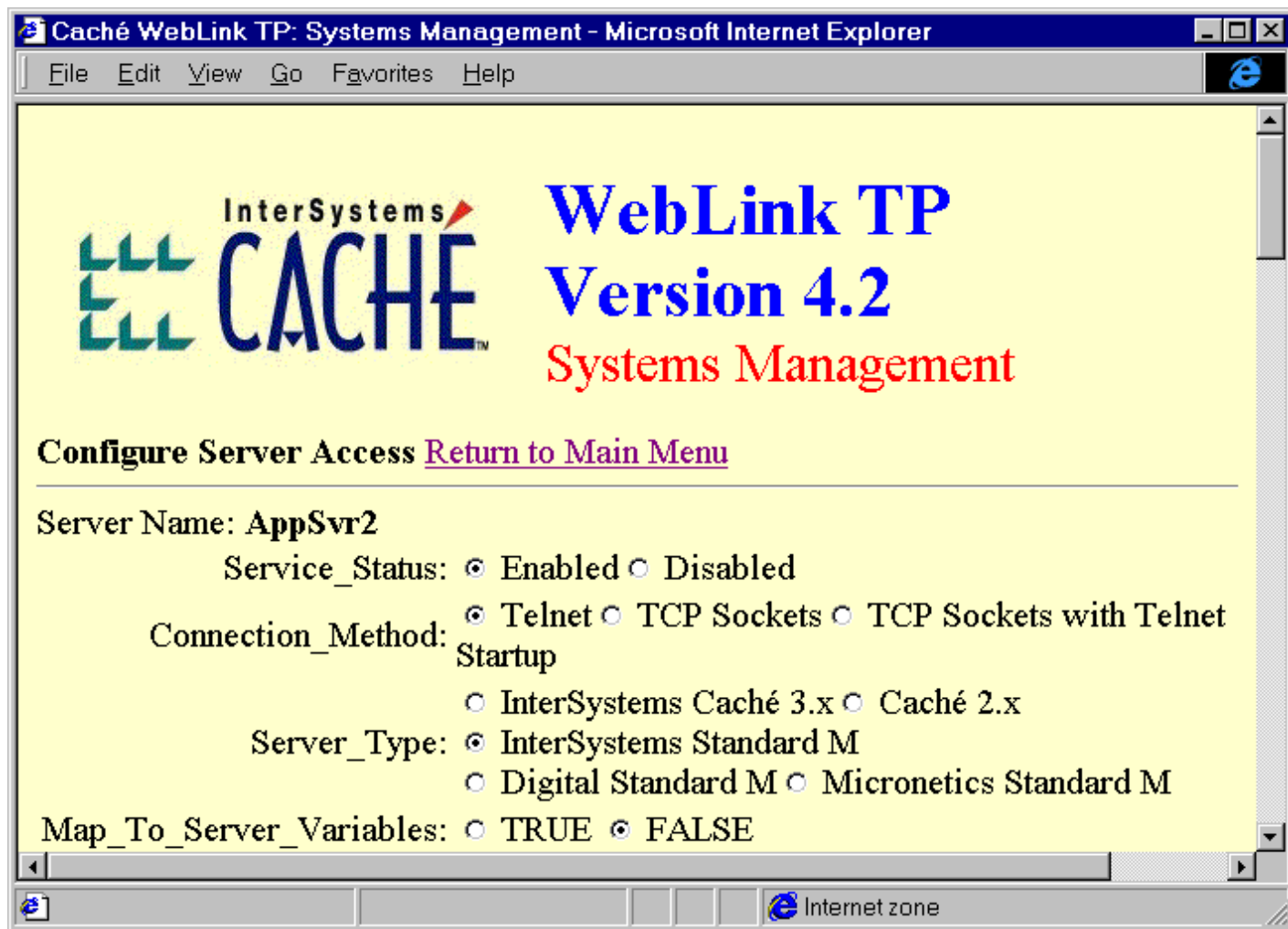
Each Caché or Open M server accessed by Caché WebLink must have an configuration entry in the Caché WebLink configuration file. The entry consists of a set of configuration parameters, which you can think of as a *server-specific configuration*. Any unspecified [optional parameters](#) and/or custom system forms are automatically inherited from the Caché WebLink [global configuration](#).

To configure access to a server:

1. From the Caché WebLink Systems Management page, select **Configure Server Access**. The following screen appears.



2. In the **Server Name** textbox, enter a unique, descriptive name for the server. This logical name is used to identify each server configuration in the Caché WebLink configuration file. It is also used as the value of the **MGWLPN** key.
3. Click the **Configure Server** button. The second configuration screen appears. Note that many parameter fields have default settings.



4. Enter the system parameters (described below) for this server configuration.

5. When you have finished, click the **Save This Configuration** button.

## Server Configuration Parameters

The server configuration form contains the fields described below.

**Note:** If you do not the State-Aware version of Caché WebLink, the options marked "Caché WebLink TP only" appear on the form, but are not enabled and therefore unusable.

### Base Parameters

#### Service\_Status

Allows you to enable and disable server configurations (default is Enabled).

#### Connection\_Method [Caché WebLink TP only]

Allows you to choose the type of TCP connection for your Caché WebLink session. Current options are:

- o Telnet
- o TCP\_Sockets (default)
- o TCP\_Sockets with Telnet Startup

#### Server\_Type

Type of InterSystems database server:

- o InterSystems Caché 3.x (default)
- o InterSystems Caché 2.x
- o InterSystems Standard M (ISM)
- o Digital Standard M (DSM)
- o Micronetics Standard M (MSM)

#### Notes:

If the Server\_Type parameter does not match the actual database server type, Caché WebLink returns an error to the client.

Windows 95/98 Caché servers and any server using 2-byte character sets require TCP socket connections.

#### Map\_To\_Server\_Variables

If set to TRUE, Caché WebLink maps URL arguments to Caché variables.

For example, if your database has the variables "name" and "id" you could send a URL like this:

`http://127.0.0.1/scripts/mgwms32.dll?name=Bob&id=21435&d-birth=12.23.71`

The information *Bob* would be passed to the **name** variable and *21435* to the **id** variable. Because **d-birth** is not a valid format for a Caché local variable, that information would be passed to the %KEY array.

If set to FALSE (the default), all arguments sent in a URL are stored in the %KEY array.

#### Slow\_Send

If set to TRUE, Caché WebLink sends data to the server in small installments: one 128-byte packet every 500 milliseconds. The TRUE setting is required only if there are telnet implementation limitation with the server. Contact your InterSystems vendor to see if this applies to your system. The default is FALSE.

#### Maximum\_String\_Length

This is the maximum string length acceptable to the target Caché system. The default value is 256 characters. Caché is capable of accepting much higher values - a value of 2048 characters would be appropriate for Caché systems.

#### Ip\_Address

The physical IP address of the server.

### TELNET Parameters

#### Login\_Profile

Specifies the login procedure for the server. If you do not specify a Login\_Profile, the user is prompted for a login.

**Note:** If you do not know the login sequence for the server, do not fill in this section. Instead, click the **Generate** button when configuring server access. For information on using the login generation form, see [Express Configuration](#).

The Login\_Profile takes the following general form:

```
username: ~ user's login name; password ~ user's password; [namespace] > ~
```

Caché systems require the namespace name.

**Example 1:** For a user named *jane* with a password of *omega*, the login profile would look like:

```
username~jane;password~omega;>~
```

**Example 2:** Using a Caché system: If you had a user named *bob* with a password of *bobbinG*, who logged into the *user* namespace, the login profile would look like:

```
username~bob;password~bobbinG;user>~
```

If you do not want to specify the password explicitly in the Caché WebLink configuration file (**mgw.ini**), you can replace the value with the reserved word **MGWPWD**. Caché WebLink substitutes the password supplied by the user for the variable **MGWPWD**.

For example, if the username is *bob*, you would specify:

```
username:~bob;password~MGWPWD;>~
```

bob would then be prompted for a password when he connected to the server.

**Caution:** Do not use any form of auto terminal detection on the target server. Configure terminal sessions for 8-bit raw mode so that character interrupts and backspaces are not processed by the telnet client. For **OpenVMS**, you need to set the passthrough option on with a DCL command. The command might look something like this:

```
$ setterm/perm/eight/pasthru/host/nowrap
```

It is recommended that you the login simple. For example, do not source the message of day, fortune files, current users, terminal auto-detection, and other such items of interest. For information on disabling any of these items, consult the system documentation for the target server.

### TCP Socket Parameters

#### Base\_TCP\_Port [Caché WebLink TP only]

Caché WebLink will allocate the port range corresponding to the starting port plus the number of licensed users. For instance, if you have 32 licensed users and your starting port is 1972, Caché WebLink will allocate ports 1972-2005; 1 port for the PCD and 32 licensed user ports.

#### PCD\_NameSpace [Caché WebLink TP only]

Specifies the namespace or UCI where the Process Control Daemon (PCD) runs.

#### Default\_User\_NameSpace [Caché WebLink TP only]

Specifies the namespace or UCI the Web clients access. This client's access will affect the data the client can see and the actions the client can perform.

### State-Aware Parameters

#### Client\_Response\_Timeout

Specifies the amount of time before an idle state-aware session will automatically close itself down. This is an additional safeguard against state-aware connections being left in use. If the target server session receives no input from the client within the timeout period, Caché WebLink will close that state-aware connection. You can also set a timeout using the key word **MGWTCO** within an HTML form or Caché ObjectScript routine, as shown in the [Closing a State-Aware Session](#) section.

## Caché 3.x Parameters

### NLS\_Translation

Specifies the translation table being used by the Caché database. The Translation table name is specified in the Caché National Language Support utility. Consult your Caché documentation for information on built-in tables, or you may use any tables you design.

## Optional Parameters

### Login\_Response\_Timeout Server\_Response\_Timeout No\_Activity\_Timeout Env\_Parameters

The descriptions of these parameters are the same as described in the [Global Configuration](#) section. If any of these parameters is blank, its value is inherited from the Caché WebLink global configuration.

### Session\_Reuse

Allows you to keep Caché Web sessions for an individual target server open to receive requests during a Web session. This option is ignored for state-aware connections.

This value is INHERIT, TRUE, or FALSE.

If set to INHERIT (the default), Caché WebLink uses the global configuration setting.

For definitions of the TRUE and FALSE values, see the [global Session\\_Reuse](#) parameter.

### Insert\_HTTP\_Headers

Allows WebLink to insert a minimalistic HTTP header in all documents not already containing an HTTP header.

This value is INHERIT, TRUE, or FALSE. If set to INHERIT (the default), Caché WebLink uses the global configuration setting.

For details on this parameter, see the [global Insert\\_HTTP\\_Headers](#) parameter.

## Custom Error Messages

### Custom System Forms

These allow you to customize WebLink responses. They are inherited from global configuration if not specified. For more information on the function of each item, see the [Customizing Error Messages](#) section.

## Copying a Server Configuration

You can quickly configure a new server by copying the configuration entry of an existing server. At that point, both configuration entries will be identical, except for the server name. You can then edit the second configuration and make changes to it (such as changing the IP address).

This feature is also useful for fine-tuning a configuration. By creating a second (temporary) configuration for a server, you can test parameter changes without worrying about losing the original configuration.

To copy an existing server configuration:

1. From the Caché WebLink Systems Management page, select **Configure Server Access**.
2. At the **Configure Server Access** screen, click on an existing server name. The second configuration screen appears.
3. In the **New Server Name** textbox, enter a unique, descriptive name for the new server.
4. Click the **Copy** button.

## Disabling Access to a Configured Server

To prevent access to a configured server, you can disable that server's configuration. The configuration will then be considered inactive, and users will not be able to access that server via this Caché WebLink system.

To disable access to a server:

1. From the Caché WebLink Systems Management page, select **Configure Server Access**.
2. At the **Configure Server Access** screen, click on a server name. The second configuration screen appears.
3. For the Server\_Status parameter, select **Disabled**.
4. Click the **Save This Configuration** button.

To re-enable access, repeat the procedure, except that you select **Enabled** at Step 3.

## Deleting a Server Configuration

To delete a configured server:

1. From the Caché WebLink Systems Management page, select **Configure Server Access**.
2. At the **Configure Server Access** screen, click on a server name. The second configuration screen appears.

3. Click the **Delete This Configuration** button.

4. Caché WebLink asks you to confirm deletion:

- To delete the configuration, click **YES:DELETE**.
- To cancel the deletion, click **Return to Main Menu**.



# Managing State-Aware Sessions

- [Opening and Maintaining State-Aware Sessions](#)
- [Using Caché Routines with State-Aware Sessions](#)
- [Using Framesets with State-Aware Sessions](#)
- [Closing a State-Aware Session](#)

---

State-Aware sessions allow a tight coupling of the Web browser and the Caché system. The browser 'owns' its own 'private' Caché process. This mechanism provides the framework for transaction processing, persistent server-side variables and true flow-control within your web application code.

## Opening and Maintaining State-Aware Sessions

Opening a state-aware connection requires setting **MGWCHD** to p.

**Example 1:** Opening a state aware session automatically.

```
<INPUT TYPE=HIDDEN NAME="MGWCHD" VALUE="p">
```

**Example 2:** Allowing the user to check a box for a state-aware session.

```
<P><INPUT TYPE=CHECKBOX NAME="MGWCHD" VALUE="p"> Start a Private State-Aware Session</P>
```

Maintaining the state-aware connection requires passing the assigned value of **MGWCHD** to the Web server by embedding a command, like the one in the example 3, in the forms the user submits.

**Example 3:** Maintaining a state-aware connection.

```
<INPUT TYPE=HIDDEN NAME="MGWCHD" VALUE=" ", $G (%KEY("MGWCHD")), " ">
```

## Using Caché Routines with State-Aware Sessions

Using the **SEND** method, you can create routines containing more than one dynamic Web page which are tied to a specifically identified user. By calling SEND (using the Caché command **Do send^%mgw**), you can send information to the Web browser and wait for the user's response.

**Example:** This example shows a routine which opens a state-aware connection, uses three discrete Web pages, each with the same header and footer, and closes the state-aware connection.

```
%TEST      ; DEMONSTRATION OF THE SEND METHOD ; 17 JUL 98 4:34 PM
           ;
A0         ; Start
           S %CR=$C(13,10)
           S ERROR=" "
           ;
FORM1      ; First form
           S NO=1 D TOP
           W "<P>Patient Number: <INPUT TYPE=TEXT NAME="NUM" SIZE="20" MAXLENGTH="20">",<CR
           D TAIL
           D send^%mgw I $D(%KEY("ESCAPE")) D LOGOUT G FORM1 ; Logout aborted
           S NUM=$G(%KEY("NUM")) I NUM="" S ERROR="You must enter the patient number" G FORM1
FORM2      ; Form 2
           S NO=2 D TOP
           W "<P>Patient Number = ",NUM
           W "<P>Patient Name: <INPUT TYPE=TEXT NAME="NAME" SIZE="20",MAXLENGTH="20">",<CR
           D TAIL
```



```

D send^%mgw I $D(%KEY("ESCAPE")) G FORM1
S NAME=%G(%KEY("NAME")) I NAME=" " S ERROR="You must enter the patient name"
G FORM2
FORM3 ; Form 3
S NO=3 D TOP
W "<P>Patient Number = ",NUM,%CR
W "<P>Patient Name = ",NAME,%CR
W "<P><B>File Data ??</B>",%CR
D TAIL
D send^%mgw I $D(%KEY("ESCAPE")) G FORM2
; File all the data here and get the next patient
G FORM1
;
TOP ; Top of form
;
S CRLF=$C(13,10)
W %CGIEVAR("SERVER_PROTOCOL"), " 200 OK",CRLF
W "Server: ",%CGIEVAR("SERVER_SOFTWARE"),CRLF
W "Date: ",$$CGIDATE($H,TZ),CRLF
W "Content-type: text/html",CRLF
W "Last-modified: ",$$CGIDATE($H,TZ),CRLF
S BROWSER=%CGIEVAR("HTTP_USER_AGENT")
I BROWSER["Mozilla",BROWSER["MSIE" W "Pragma: no-cache",CRLF
I BROWSER["Mozilla",BROWSER["MSIE"
W "Expires: ",$$CGIDATE((+$H_", "_$P($H,",",2)-5),TZ),CRLF
W "Window-target: ",TARGET,CRLF ; Netscape only
S TITLE="Form "_NO
W "<HTML>",%CR
W "<HEAD><TITLE>",TITLE,"</TITLE></HEAD>",%CR
W "<BODY>",%CR
W "<FORM METHOD=POST ACTION=\"/scripts/mgwms32.dll /\">",%CR
W "<INPUT TYPE=HIDDEN NAME=\"MGWCHD\" VALUE=\"\",$G(%KEY(\"MGWCHD\")),\"\">",%CR
W "<H2>",TITLE,"</H2>",%CR
W "<P><A HREF=\"/scripts/mgwms32.dll?MGWCHD=\",$G(%KEY(\"MGWCHD\")),\"&ESCAPE=1\">"
W "Previous Form</A>",%CR
W "<HR>",%CR
Q
;
TAIL ; End of form
W "<P><INPUT TYPE=\"SUBMIT\" NAME=\"SUBMIT\" VALUE=\"Submit\">",%CR
W "<HR>",%CR
I $L(ERROR) W "<B>ERROR: ",ERROR,"</B><HR>",%CR S ERROR=" "
W "</FORM>",%CR
W "</BODY>",%CR
W "</HTML>",%CR
Q
;
LOGOUT ; Logout of system
S NO="[LOGOUT]" D TOP
W "<INPUT TYPE=HIDDEN NAME=\"MGWCDN\" VALUE=\"\",$G(%KEY(\"MGWCHD\")),\"\">",%CR
W "<P><B>Press the SUBMIT button to log out</B>",%CR
D TAIL
D send^%mgw ; We are now dead, WebLink will send the 'logged out' banner
Q
;

```

## Using FrameSets With State-Aware Sessions

WebLink automatically queues individual frame transactions such that a single Caché process can deal with them in sequence. This requires that the SEND method following a frameset document must, not only 'send' the frameset document itself, but also respond to requests for individual frames, and events triggered through individual frames.

The following sample code shows an application with framesets using the SEND method. Forms 1 and 3 are 'normal' full-screen forms. Form 2 is a frameset document in which two individual frames are specified. Notice how the frameset SEND method will return to request the actual contents of the two frames. Also, user input from either of the two frames must be handled through this control loop. The program should exit the loop when action from the user indicates that control should pass from the frameset document to the next form (Form 3).

```

FORM1 ; First form
D FORM1D ; Write out form 1

```

```

        D send^%mgw
        ; Respond to user input via form 1
FORM2    ; Second form
        D FORM2D ; Write out form 2 (a frameset document)
LOOP2    D send^%mgw
        I [Frame1 requested] D FORM2aD G LOOP2 ; Write frame1
        I [Frame2 requested] D FORM2bD G LOOP2 ; Write out frame2
        I [user input from Frame1] D FORM2aX G LOOP2 ; Respond to user input via frame1
        I [user input from Frame2] D FORM2bX G LOOP2 ; Respond to user input via frame 2
        ; If none of above, proceed to FORM 3
FORM3    ; Third Form
        D FORM3D ; Write out form 3
        D send^%mgw
        ; Respond to user input via form 3

```

## Closing a State-Aware Session

State-aware sessions must be explicitly closed in order to free up a Caché WebLink TP connection. The connection is closed in one of three ways:

- The Caché WebLink system manager [closing the connection](#).
- The web application closing the connection (as shown in examples 1 and 2).
- The connection expiring. The connection expiration is set with a configuration parameter called [Client\\_Response\\_Timeout](#), or with a WebLink reserved variable **MGWCTO** (as shown in example 3).

The programmer can set the web application to explicitly close a state-aware session by setting the value of **MGWCDN**, the call to terminate a Caché session, to the value of **MGWCHD**, the connection handle. The programmer can also redirect the browser to a state-less session after closing the state-aware session by including the value of the next Caché service as shown in example 2.

### Example 1: Explicitly closing a state-aware Caché session

```

W "<INPUT TYPE=HIDDEN NAME=" "MGWCHD" VALUE=" ", $G(%KEY("MGWCHD")), " ">", $C(13,10)
W "<INPUT TYPE=HIDDEN NAME=" "MGWCDN" VALUE=" ", $G(%KEY("MGWCHD")), " ">", $C(13,10)

```

**Example 2:** Closing a state-aware Caché session and redirecting the browser to a stateless session connected to Server\_A. You can also include a password if needed.

```

W "<INPUT TYPE=HIDDEN NAME=" "MGWCHD" VALUE=" ", $G(%KEY("MGWCHD")), " ">", $C(13,10)
W "<INPUT TYPE=HIDDEN NAME=" "MGWCDN" VALUE=" ", $G(%KEY("MGWCHD")), " ">", $C(13,10)
W "<INPUT TYPE=HIDDEN NAME=" "MGWLPN" VALUE=" "Server_A" ">", $C(13,10)

```

**Note:** This is a simplistic example with minimum information required. You may want (or need) to pass more information back to the Web server.

**Example 3:** Setting **MGWCTO** to time-out in 2 minutes within a routine. This value supersedes the value set for [Client\\_Response\\_Timeout](#) in the Caché system configuration.

```

W "<INPUT TYPE=HIDDEN NAME=" "MGWCTO" VALUE=" "120" ">", $C(13,10)

```

The code shown in example 3 is sufficient to time-out a connection; however, this may leave unfinished transactions. To combat this issue, you can invoke a routine to clean up any unfinished transactions. Caché WebLink provides two extra parameters that invoke a routine which only runs if the client idle time exceeds the client response time-out setting: **MGWCTOproc**, used within an HTML page or hyperlink as shown in example 4; and **MGWCTOflag**, used within the back-end Caché routine, as shown in example 5.

**Example 4:** Invoking a local routine called "CLEANUP" if the client (browser) times-out, using a form. The example sets the client response time-out to 120 seconds. Caché WebLink obtains the connection handle, which it uses to reconnect the client to its private session, from the %KEY array.

```

W "<FORM METHOD=POST ACTION=" "/scripts/mgwms32.dll
?MGWCHD=" " ", $G(%KEY("MGWCHD")), " "&MGWCTO=120&MGWCTOproc=" "^CLEANUP" ">", $C(13,10)

```

**Note:** There should be no spaces in the URL.

This disadvantage to the method shown in example 4 is that it exposes a Caché routine name to the user. A potentially better way to handle invoking a rollback procedure for unfinished transactions uses **MGWCTOflag** as shown in example 5.

**Example 5:** Invoking a local routine called "CLEANUP" if the client (browser) times-out. This example shows how this mechanism would be used with the SEND method.

```
D send^%mgw I $D(%KEY("MGWCTOflag")) D CLEANUP H
```

The variable **MGWCTOflag** only exists if the [Client\\_Response\\_Timeout](#) or the value set in **MGWCTO** in the client's previous response has been exceeded. It does not exist if the browser responds within the specified time-out period. The private Caché session halts after the clean-up routine completes. If the browser subsequently tries to connect to the private session, Caché WebLink will send an error page. The user must open a new session, to redo the transaction from the beginning.

**Example 6:** Relying on a generic time-out procedure. If no specific action is defined for when a state-aware session times-out, WebLink will attempt to transfer control to a procedure defined in:

```
^MGWAPP("TIMEOUT")=<routine_name>
```

If a 'time-out' procedure is not defined here, WebLink will attempt to run a procedure named **TIMEOUT** in your application launch routine (i.e. %MGW2 or %ZMGW2).



# Manipulating Queries

- [The Caché WebLink System: Software Components](#)
  - [URL-based Web-to-Caché Queries: The 'GET' method](#)
  - [Submitting HTML Forms to Caché: The 'POST' method](#)
  - [Processing HTML Forms and URLs in Caché](#)
  - [Reserved Key Names in HTML Forms](#)
  - [CGI Environment Variables](#)
  - [Using Cookies in WebLink](#)
  - [Troubleshooting](#)
- 

Users send queries to Caché or Open M databases through Caché WebLink by means of URLs and standard Web pages. Examples of these Web pages are provided by the HTML forms used in configuring Caché WebLink. An HTML form or hyperlink can transmit information to the Web server which, in turn, passes it (using Caché WebLink) to your Caché applications. Your application, using the information received, generates a response, as an HTML form, which is returned to WebLink which, in turn, passes it on to the user via the Web server. This section discusses how to set up and submit queries and receive the responses generated by Caché.

## The Caché WebLink System: Software Components

Caché WebLink is implemented as a single library which resides on the Web server's host machine. This library is responsible for maintaining the dialogue between the Web server and Caché. Depending on your hosting environment, WebLink's library will be one of the following:

### **mgwms32.dll**

For Microsoft-based Web servers.

### **mgwns20.dll or mgwns30.dll**

For Netscape-based Web servers operating under Microsoft NT, Windows 95, or Windows 98. Use the '20' library for Netscape Web servers version 2.x and the '30' library for Netscape Web servers version 3.x.

### **mgwns20.so or mgwns20.sl or mgwns30.so or mgwns30.sl**

For Netscape-based Web servers operating under UNIX. Depending on the version of UNIX you are using, the library extension will be either '.so' (shared object) or '.sl' (shared library). As with the Microsoft platform, use the '20' library for Netscape Web servers version 2.x and the '30' library for Netscape Web servers version 3.x.

The Caché WebLink library communicates with the following routines on the Caché server:

### **%mgw, %mgw0, %mgwa, %mgwb, %mgwd and %mgwj**

These routines make up the Caché WebLink Daemon.

### **%mgwe**

This is the Caché-side Event and Error log.

### **%mgw1**

This routine is required for Caché/NextGen systems only.

### **%mgw3**

This routine contains a built-in test response HTML page - it is a template for application launch routines %MGW2 (or %ZMGW2). If you elect to use the routine-based method to launch your application code (as opposed to the global-based method) %mgw3 will be called if neither %MGW2 nor %ZMGW2 is present.

### **%MGW2 or %ZMGW2**

This routine is called by WebLink's Daemon if you are using the routine-based method to launch your application code. Launch your Caché application code from this routine. Caché systems should use %ZMGW2 - this routine will be protected when Caché is upgraded. If you don't have either routine, copy routine %mgw3 to %MGW2 (or %ZMGW2).

### **Optional: %MGW and %MGWJ**

Older versions of WebLink (pre-version 4.2) used upper-case routines for the Daemon. If your application code is still calling upper-case names, you must install these two 'stub' routines for backwards compatibility.

**Note:** The user must have access to read and execute the Caché WebLink routines and the routines you create to run your application on the individual Caché systems. Also, all WebLink processes (in all NameSpaces) must be able to read and write to global ^%MGW.

## URL-based Web-to-Caché Queries: The 'GET' method

Users can submit queries to Caché by means of information appended to the URL identifying WebLink. Information is submitted to

Caché as pairs of Keys and Values. Think of each Key as a variable name which has an associated value. Queries submitted in this way are said to use the HTTP 'GET' method.

The basic structure of the complete URL is as follows:

`http://<host>/scripts/mgwms32.dll?<Key>=<Value>[&<Key>=<Value>]`

The following table describes the syntactical elements within the URL:

Format of a URL with an embedded Cachéquery	
Element	Definition
<code>http://&lt;host&gt;/scripts/mgwms32.dll</code>	The URL for the site name and the Caché WebLink library.
<code>?</code>	The separator between the URL and the key parameters passed to the Caché system.
Key	The name of the key being passed.
<code>=</code>	The separator between the key name and the key value.
Value	The value of the key being passed.
<code>&amp;</code>	The separator between two keys.

**Note:** Web page tags are not case sensitive. Caché routines, key names and values are case sensitive.

Certain key names are reserved in WebLink; for example, those which identify the target Caché system.

## Specifying the Target Caché System

Reserved key name **MGWLPN** always identifies the Caché system to which the query should be submitted. The value of **MGWLPN** should be the name of the target Caché system as defined within WebLink's configuration. For WebLink-to-Caché communications via telnet, a password for the telnet service may be required - this should be submitted as the value of reserved key name **MGWPWD**.

Every query submitted through WebLink should identify the Caché system (**MGWLPN**) and, if necessary, provide the password (**MGWPWD**). WebLink will generate a '**connection handle**' (**MGWCHD**) based on the identified Caché system. The value of **MGWCHD** alone can be used to identify the Caché system in subsequent requests. **MGWCHD** is also used to maintain state-aware communications within WebLink applications.

Examples of WebLink URLs are given below:

**Example 1:** This example illustrates a simple Web-to-Caché query. The target Caché system is defined, a password for the telnet service is supplied and a user-defined key (MyKey) is included:

`http://127.0.0.1/scripts/mgwms32.dll?MGWLPN=System_A&MGWPWD=MyPassword&MyKey=MyData`

**Example 2:** This example illustrates how state-aware sessions are started in WebLink. Again, some user-defined keys are shown in addition to WebLink's reserved keys:

`http://127.0.0.1/scripts/mgwms32.dll?MGWLPN=System_A&MGWCHD=p&MyKey1=MyData1&MyKey2=MyData2`

## Submitting HTML Forms to Caché: The 'POST' method

This section deals with the case where the entire contents of an HTML form are submitted to Caché. Such submissions employ the HTTP 'POST' method.

The following example shows a form which allows the user to identify and connect to an Caché system. The following points are illustrated:

- Constructing the **POST** and **ACTION** statement to direct the form's data to Weblink when the user submits it (by pressing the **SUBMIT** button).
- Allowing the user to select from a choice of Caché systems (System\_A, System\_B, and System\_C, in this example).
- Allowing the user to enter a password for the chosen Caché system.
- Using reserved key names, in this case **MGWPWD** and **MGWLPN** to identify the target Caché system.
- User-defined Key/Value pairs are included - a **TEXT** element for the user's name and a **TEXTAREA** element to enable the user to add free text are shown.

Caché Weblink calls **mgwms32.dll** to process this form.

```
<HTML>
<HEAD>
<TITLE>Caché Weblink: Front Page</TITLE>
</HEAD>
<BODY>
<H2>Caché Weblink: Universal HTML Service for Caché</H2>
<P>Login to a Caché System</P>
<FORM METHOD=POST ACTION="/scripts/mgwms32.dll">
<P>Server Name: <SELECT NAME="MGWLPN" MULTIPLE>
<OPTION SELECTED>System_A<OPTION>System_B<OPTION>System_C
</SELECT>
Password:<INPUT TYPE=PASSWORD NAME="MGWPWD" SIZE="20" MAXLENGTH="20">
<P></P>
Your Name:<INPUT TYPE=TEXT NAME="Name" SIZE="20" MAXLENGTH="20">
<P></P>
<TEXTAREA NAME="Comments" ROWS="6" COLS="40" WRAP="Virtual"></TEXTAREA>
<INPUT TYPE=SUBMIT NAME="Login" VALUE="Connect to System">
</FORM>
<HR>
</BODY>
</HTML>
```

This form establishes a dialog with a Caché system. The Caché system can sustain communication with the user by generating HTML pages in response to user (client) input.

## Processing HTML Forms and URLs in Caché

When a client submits an HTML form or URL, Caché Weblink extracts all key names and their values from the form/URL and sends them to the Caché system (a Weblink Daemon process) along with the CGI environment variables relevant to the transaction.

The Weblink Daemon calls and passes all submitted key/value pairs and CGI environment variables to your application code which should then process and respond to the user's query by generating an appropriate HTML form.

The following procedures occur when a form/URL is submitted to Caché:

1. [Receiving the form's data into the Caché environment](#)
2. [Launching your application code to process the user's query](#)
3. [Sending the HTML response to the user](#)

### Receiving the Form's Data Into the Caché Environment

Caché Weblink passes the form/URLs data to your application code in two arrays: **%KEY** and **%CGIEVAR**. These arrays contain all the information you need to process the user's query. The request's method name (**GET** or **POST**) is passed to your code in Caché variable **%METHOD**

As a programmer, you have complete control over what information is passed from the user to the Caché system.

**Note:** A Web client using Caché Weblink to access Caché databases has no direct access to the Caché environment. Web clients cannot execute Caché commands.

## **%KEY(NAME)=VALUE**

This array contains the values the user entered into the form fields. These values may be very large or may include multiple options. The amount of data that can be accepted as a single string is controlled by the configuration parameter [Maximum\\_String\\_Length](#). This parameter represents the total number of characters that can be held in a single string - it should be specified for each Caché system defined within Caché WebLink. The default value for the 'Maximum\_String\_Length' is 256 characters. Modern Caché systems are capable of processing much longer strings. For example, a value of 2048 characters would be more appropriate for Caché systems. For the purpose of this discussion we will use the default value of 256 characters.

If the value of any key variable exceeds 256 characters, Caché WebLink will automatically divide it into an array with each subscript containing no more than 256 characters. If the user is presented with choice of options where any number of options, not just one, can be selected, Caché WebLink will read the submitted options into a subscripted array. The examples below are based on the code from the previous section, [Submitting HTML Forms to Caché](#).

### **Sample Code for examples below**

```
<FORM METHOD=POST ACTION="/scripts/mgwms32.dll">
<P>Server Name: <SELECT NAME="MGWLPN" MULTIPLE>
<OPTION SELECTED>System_A<OPTION>System_B<OPTION>System_C
</SELECT>
Password: <INPUT TYPE=PASSWORD NAME="MGWPWD" SIZE="20" MAXLENGTH="20"><P></P>
<P></P>
Your Name:<INPUT TYPE=TEXT NAME="Name" SIZE="20" MAXLENGTH="20">
<TEXTAREA NAME="Comments" ROWS="6" COLS="40" WRAP="Virtual"></TEXTAREA>
<P>
<INPUT TYPE=CHECKBOX NAME="MGWCHD" VALUE="p"> Start a Private State-Aware Session></P>
<INPUT TYPE=SUBMIT NAME="Login" VALUE="Connect to System">
</FORM>
```

**Note:** The **MGWCHD** variable in this example is used to create a state-aware session.

### **Example 1: A simple assignment**

In the above code sample, Caché WebLink requests a password using the `<INPUT>` tag , Caché WebLink interprets the variable input in this form:

```
%KEY( "MGWPWD" )=passwordvalue
```

### **Example 2: A form element submitting a large value**

If the value exceeds 256 characters, Caché WebLink automatically divides it into an array with each subscript containing no more than 256 characters. In the above code sample, the `<TEXTAREA>` field may have a value larger than 256 characters. Caché WebLink will read long data values as follows:

```
%KEY( "Comments",0,1)="The first 256 characters"
%KEY( "Comments",0,2)="The second 256 characters"
```

WebLink will continue adding subscripted sections to the array until all data for the form's element has been read.

### **Example 3: A form element containing multiple values**

In the above code sample, Caché WebLink requests a server name using the `<SELECT>` tag.

If the user selects one option, for example `System_B`, WebLink will present this data as follows:

```
%KEY( "MGWLPN" )="System_B"
```

If, however, the user selects two options, for example, System\_A and System\_C, WebLink will present both values in the array by assigning each value its own subscript:

```
%KEY( "MGWLPN",1)="System_A"  
%KEY( "MGWLPN",2)="System_C"
```

It should be noted that this is a contrived example - a form can only be submitted to one Cachésystem at a time.

#### Example 4: A form element containing multiple long values

If you have a situation where your user has a choice of long strings (not shown in the code sample), Caché WebLink will present the data submitted as follows:

```
%KEY( "Comments",1,0,1)="The first 256 characters of the first option"  
%KEY( "Comments",1,0,2)="The second 256 characters of the first option"  
%KEY( "Comments",2,0,1)="The first 256 characters of the second option"  
%KEY( "Comments",2,0,2)="The second 256 characters of the second option"  
%KEY( "Comments",3)="The third option which is less than 256 characters"
```

- continuing until all information has been assigned.

#### Example 5: Form data is too large to fit into local storage

There are situations where the total size of the form's data becomes too large to be accommodated in the %KEY local array. For example, 'TEXTAREA' elements can contain many lines of written text.

WebLink provides the facility to read a form's data into permanent storage (i.e. a global) instead of a local array. The need to use this facility must be planned for in advance - the form must submit the reserved variable '**MGWGLO**' to instruct WebLink to read the form's data into a global.

We could define key '**MGWGLO**' in our sample code by adding the following line:

```
<INPUT TYPE="HIDDEN" NAME="MGWGLO" VALUE="1">
```

The structure of the '%KEY' global is as follows:

```
^MGW( "%KEY", $J,
```

The structure beneath this level of subscripting is the same as that for the %KEY array.

The WebLink system variable **%ZCS("GLO")** will be set to **TRUE** if the incoming form's data has been read into a global.

```
if '$g(%ZCS("GLO")) set Name=$g(%KEY("Name"))  
if $g(%ZCS("GLO")) set Name=$g(^MGW("%KEY", $J, "Name"))
```

Note that CGI environment variables will always be written to an array (%CGIEVAR) irrespective of the status of 'MGWGLO'.

Note also that if global storage is requested, WebLink will continue to store its reserved key variables (i.e. those whose names begin with 'MGW') in the %KEY array, in addition to placing them in the global. Of course, all other form variables will not be duplicated in this way.

PDQweb applications also have access to this facility. Their form variables will be stored in:

```
^MGW( "%", $J,
```



The structure of PDQweb's '%' array will be preserved below this level of subscripting.

### **%CGIEVAR(NAME)=VALUE**

This array holds CGI environment variables and their associated values for each transaction. The Web server passes these variables automatically. They include information such as the IP address of the user and the type of browser the user is using. The standard CGI variables are defined in most HTML and CGI reference materials. Caché WebLink provides the variables you are most likely to need.

For example, the remote user's IP address is held in the variable %CGIEVAR( "REMOTE\_ADDR" )

See the section on [CGI Environment Variables](#) for further information.

## **Launching Your Application Code to Process the User's Query**

You must create a Caché routine (or routines) which will process the user's query and return an HTML form in response. There are two methods for launching WebLink application code:

1. Through the routines %MGW2 (or %ZMGW2).
2. Through the global ^MGWAPP.

### **Launching Your Application Code From Routine %MGW2 or %ZMGW2**

Users of Caché should embed calls to their application code in routine %ZMGW2, not %MGW2. %ZMGW2 will be protected when the Caché system is subsequently upgraded.

The launch routine must be able to process and differentiate amongst the responses received from the HTML forms and menus that make up each Web-enabled application on your Caché system. Each form/URL is identified by the Key values it contains. For example a typical entry in %MGW2 (or %ZMGW2) would look like:

```
HTML ; Process incoming HTML data
      if $d(%KEY("MyWebApp")), $d(%KEY("form1")) do form1^MyWebApp quit
      if $d(%KEY("MyWebApp")), $d(%KEY("form2")) do form2^MyWebApp quit
```

### **Launching Your Application Code from Global ^MGWAPP**

References to your application code can be stored in the global ^MGWAPP, as an alternative to the routine-based approach previously described.

Use the reserved key variables **MGWAPP**, **MGWAPP1**, **MGWAPP2** and **MGWAPP3** to identify the appropriate routine contained within ^MGWAPP.

#### **Example 1:**

Form Variable:

```
... &MGWAPP=HomePage
```

will identify the following global node:

```
^MGWAPP( "HomePage" ) = "Home^MyApp"
```

The submitted query will be processed by routine:

```
Home^MyApp
```

### Example 2:

Form Variables:

```
...&MGWAPP=Customers&MGWAPP1=Orders_Form
```

will identify the following global node:

```
^MGWAPP("Customers","Orders_Form")="ORD^CUST"
```

The submitted query will be processed by routine:

```
ORD^CUST
```

In all cases, WebLink will transfer control to the launch routine %MGW2 (or %ZMGW2) if it fails to find a suitable procedure in global ^MGWAPP.

## Sending the HTML Response to the User

Send lines of HTML code back to the Web server (via WebLink). Terminate each line with carriage-return (ASCII 13) and line-feed (ASCII 10). It is particularly important that each HTTP header is terminated with a carriage-return and line-feed. HTTP headers must be terminated with an extra carriage-return line-feed before proceeding with the HTML code. Note that WebLink can be configured to automatically add HTTP headers to your HTML responses - see the [Insert\\_HTTP\\_Headers](#) configuration parameter for further information.

In Caché software, carriage-return, line-feed may be written as `$c(13,10)`.

**Warning:** The TCP Sockets protocol will not accept the abbreviation "!" for carriage-return, line-feed.

**Tip:** When sending a large amount of data back to the client, regularly flush Caché's output buffer. This is done by calling the **flush^%mgw** procedure in your application code. Caché will automatically flush its output buffer when it is full (usually 32 KB), but other Open M products rely on your application code to do this and will return an error if the output buffer size is exceeded. Flushing the output buffer regularly will result in your Web application appearing to be more responsive to the user.

The following example shows Caché code returning an HTML response to the user. Note the use of the WRITE command to output HTML code and HTTP headers. This example uses the Netscape-specific extension Window-target to direct the output to the main window.

### Example:

```
WRITE "HTTP/1.0 200 OK",$c(13,10)
WRITE "Content-type: text/html",$c(13,10)
WRITE "Window-target: main_window:,$c(13,10)
WRITE $c(13,10)
WRITE "<HTML><HEAD><TITLE>Sample Web Page </TITLE></HEAD>",$c(13,10)
WRITE "<BODY>",$c(13,10)
WRITE "<P>These are the results of your query...</P></BODY></HTML>",$c(13,10)
```

## Sending Individual Lines of an HTML Page from Caché to the Web Server

There are two methods for sending lines of HTML form data back to the client via the Web server.

- Using the procedure **ss^%mgw**:  
Set each line in variable %TXT and call **ss^%mgw**. Alternatively, set many lines in a numbered array %TXT and call **ss^%mgw**.  
For example:

```
SET %TXT(1)="<HTML>"
SET %TXT(2)="<HEAD><TITLE>The Big Hospital</TITLE></HEAD>"
SET %TXT(3)="<BODY><H2>Admissions Data</H2></BODY></HTML>"
DO ss^%mgw
```

- Using WRITE statements:  
Write each line as if you were simply writing to the current device. Include %C(13,10) to insert a carriage-return/line-feed pair. If you use this method be sure to use WRITE %ZCS("DRW") or WRITE %C(13,10) to terminate your response

A form generated in response to the form submitted to Caché in the section [Submitting HTML Forms to Caché](#) might be:

```
<HTML>
<HEAD><TITLE>The Big Hospital</TITLE></HEAD>
<BODY><H2>Admissions Database</H2>
<FORM METHOD=POST ACTION="/scripts/mgwms32.dll">
<INPUT TYPE=HIDDEN NAME="MGWLPN" VALUE="System_A"
<INPUT TYPE=HIDDEN NAME="MGWCHD" VALUE=" ", $G(%KEY("MGWCHD")), "">
<A HREF="/scripts/mgwms32.dll?MGWCHD=", $G(%KEY("MGWCHD")), &ESCAPE=1 "">
<P>Enter Patient Number: <INPUT TYPE=TEXT NAME="PAT-NUM" SIZE="8" MAXLENGTH="8">
<P>Browse: <INPUT TYPE=RADIO NAME="RB" VALUE="REG" CHECKED>Registration<INPUT TYPE=RADIO NAME="RB"
VALUE="ADM">Admissions<INPUT TYPE=RADIO NAME="RB" VALUE="HAEM">Haematology<INPUT TYPE=RADIO NAME="RB"
VALUE="CHEM">Chemistry<P>
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit Query">
<INPUT TYPE=RESET>
</Form></BODY></HTML>
```

Use 'hidden' fields to send the keys **MGWLPN** and **MGWCHD** back to the client silently. If a password, **MGWPWD**, was supplied initially, it can be returned to the client also. This procedure is important, particularly in environments consisting of many Caché configurations. The login parameters ensure that the next reply to this form is sent to the same Caché system. If, for reasons of security, you do not wish to send the explicit values of these fields back to the Web browser, then use the connection handle, **MGWCHD**, to allow Caché WebLink to identify **MGWLPN** and **MGWPWD** internally. On Caché WebLink TP sessions, **MGWCHD** also handles the state-aware flag. **MGWCHD** alone must be used to maintain the connection in state-aware Web applications.

For telnet configurations where you allow the reuse of connected sessions (**Session\_Reuse=TRUE**), never assume that Caché WebLink can reconnect you to the specific session that you opened. Always supply the login profile in some form so that Caché WebLink can log into a new session using the existing session from the Web server to the Caché system. There are circumstances where you provide the Login\_Profile implicitly and therefore do not need to include it explicitly in the configuration. Consult the chart below.

Target Caché system is the default Caché system defined	Password field in the Login_Profile contains the MGWPWD key	Explicitly include this key
Yes	No	no keys needed
Yes	Yes	MGWPWD
No	Yes	MGWLPN and MGWPWD
No	No	MGWLPN

For information about setting the default Caché system, see the [Systems Management](#) section. For information about using MGWPWD, see [Reserved Key Names in HTML forms](#).

## Reserved Key Names in HTML Forms

The following key names are reserved for internal use by Caché WebLink.

- MGWLPN**  
The target Caché system name, as defined in your Caché WebLink configuration. See the section on [Specifying the Target Caché System](#) and [Submitting HTML Forms to Caché](#) for examples on how to use this parameter.
- MGWPWD**  
This key is relevant to *telnet-based* connections only.

The password to the Caché system defined in **MGWLPN**. Set this key name to hold the password of the telnet service to be used in establishing a connection.

#### **MGWNSP** or **MGWUCI**

This key is relevant to *TCP socket-based* connections only.

Use this key to specify the NameSpace you wish to connect to on the target Caché system. Applications should use **MGWNSP**. **MGWUCI** is provided for backwards compatibility.

For *telnet-based* connections it is expected that the telnet [Login\\_Profile](#) will directly login to the appropriate NameSpace.

#### **MGWCHD**

The connection handle.

#### **stateless Connections:**

This value of this key is generated by WebLink and passed to your application. It represents the information you supplied in order to establish a connection to the Caché system: i.e. **MGWLPN**, **MGWPWD**, and **MGWNSP**. Having received this parameter in your first connection, you can use it to identify the Caché system (and NameSpace) for future queries to the same Caché system. This avoids having to repeatedly send the values of **MGWLPN**, **MGWPWD** and **MGWNSP** to the Web browser.

For example, the following line of code shows how this key would be used to identify the Caché system in a form generated by your Web application:

```
write "<INPUT TYPE='HIDDEN' NAME='MGWCHD' VALUE='$_$g(%KEY('MGWCHD'))_'">"
```

Specifying **MGWCHD** as shown above will allow WebLink to identify the Caché system when the form is subsequently submitted.

Using **MGWCHD** improves security because the actual names of the target Caché systems are disguised. Your application should never modify the value of **MGWCHD** in stateless connections.

#### **State-aware Connections:**

For state-aware connections this key not only identifies the Caché system and NameSpace as with stateless connections, but it identifies the **private** process to which you are connected.

To start a private state-aware session set **MGWCHD** to 'p' in the initial connection:

```
...&MGWCHD="p"
```

WebLink, having allocated a private session, will pass a modified version of this key to your state-aware Web application. Use this value to subsequently maintain your connection - you must not use **MGWLPN** once your private session has been established. See the section on maintaining [State-aware sessions](#) for further information.

#### **MGWLIB**

This key is automatically generated by WebLink - it contains the value of the CGI environment variable **SCRIPT\_NAME**. The value of **SCRIPT\_NAME** is the part of the URL which identifies WebLink. For conventional WebLink configurations the value of this variable will be **/scripts/mgwms32.dll**. Use this key to construct portable WebLink URLs in your Web applications. For example, the following line of code illustrates how the value of **MGWLIB** would be used to construct a **METHOD** statement within a form:

```
write "<FORM METHOD=POST ACTION='$_$g(%KEY('MGWLIB'))_'">"
```

It is recommended that you always use **MGWLIB** to construct Weblink URLs. This will ensure that your applications are portable across different Weblink configurations. For example, in Apache/CGI-based configurations, WebLink's URL would be **/cgi-bin/nph-mgwcgi** and not **/scripts/mgwms32.dll**.

#### **MGWCDN**

Use this key to terminate specified Caché sessions. If you have [Session Reuse](#) set as TRUE in the main **SYSTEM** configuration, or you use state-aware Caché sessions, you can use this key to terminate all Caché sessions periodically, or just the sessions for a particular Caché system without shutting down the Web server itself. If you set this key name to the appropriate Caché system's name, Caché WebLink will close all existing Caché sessions for that Caché system. Set this key value to "\*" to close down all existing Caché sessions on all Caché systems. Caché WebLink will only accept this variable from a system defined in the [System Manager](#) field.

For information on using MGWCDN to close state-aware sessions, see [Closing State-Aware Sessions](#).

#### MGWSTO

The amount of time the browser will wait for the Caché server to respond. This variable corresponds to the [Server Response Timeout](#) variable you set on your Caché WebLink server extension or on an individual Caché system configuration. Using this variable, you can override the settings in your Caché WebLink server configuration.

#### MGWCTO

This key is relevant to **stateless** communications only.

The client-response timeout variable allows you to set a maximum time to wait for a response from a Web browser within an application. This adds greater security for state-aware connections by not leaving idling Web browsers connected to your system and frees up the connection for another Web client. You can also set this on a per system basis in your Caché system configuration by assigning a value to the [Client Response Timeout](#) parameter.

#### MGWEBP or MGWJSP

This key is automatically generated and passed to your application by WebLink for Event Broker-enabled configurations. Its value is the port number the Event Broker is listening on. Applications should use **MGWEBP**. **MGWJSP** is provided for backwards compatibility. See the section on the [Web Event Broker](#) for further information.

#### MGWGLO

This key, if defined, will instruct WebLink to read the HTML form's data into permanent storage (a global) instead of local storage (an array). Use this facility if you anticipate a form being able to accept huge amounts of user input - for example, forms containing **TEXTAREA** elements. See the section on [Receiving the form's data into the Caché environment](#) for further information.

#### MGWCGI

Use this key to request additional CGI environment variables for your application - i.e. variables not automatically supplied by WebLink. Extra CGI environment variables are defined as a comma-separated list. See the section on [Requesting extra CGI environment variables](#) for further information.

#### MGWSCR

For efficiency, WebLink buffers HTML form data received from your Web application and periodically flushes this buffer to the browser via the Web server. In general, it is more efficient to dispatch a few large data buffers over the network than lots of small ones.

However, in some cases, it is desirable, particularly with forms which take a long time to generate to dispatch relatively small volumes of data to the browser. Doing this gives the user some visual indication that progress is being made. This key, if defined, will cause WebLink to immediately dispatch data to the browser (via the Web server) every time carriage-return/line-feed is encountered.

For example, consider the following HTML-generating code fragment:

```
for i=1:1:10 write "<P>Time elapsed: ",i,$char(13,10) do flush^%mgw hang 1
```

Assuming that you requested this form with **MGWSCR** defined, you will see individual lines appearing in the browser window at one second intervals.

For this facility to be effective, it is necessary to guard against Caché buffering output data and the browser buffering input data. Bearing this in mind, notice that:

- Each line is terminated with carriage-return, line-feed: `$char(13,10)`.
- The output buffer (from Caché) is flushed after each line (`flush^%mgw`). This is particularly important for TCP socket-based connections. If Caché's output buffer is not flushed, the mechanism will be inactive due to Caché

- buffering the output data.
- Each line constitutes a new line in the browser's window. This is achieved in the example by preceding each line with **<P>** - new paragraph (or **<BR>** - new line). This is necessary because the browser will buffer data until it knows it can format an entire line, at which point it will display the line.

## EP

This key is relevant to **PDQweb** applications only.

This key identifies the Caché procedure to run - as defined in the PDQweb application file. See the [PDQweb compatibility](#) section for further information.

## CGI Environment Variables

The following CGI environment variables are automatically passed to your Caché application with each and every request.

- AUTH\_PASSWORD
- AUTH\_TYPE
- CONTENT\_TYPE
- GATEWAY\_INTERFACE
- HTTP\_ACCEPT
- HTTP\_AUTHORIZATION
- HTTP\_COOKIE
- HTTP\_REFERER
- HTTP\_USER\_AGENT
- PATH\_TRANSLATED
- REMOTE\_ADDR
- REMOTE\_HOST
- REMOTE\_IDENT
- REMOTE\_USER
- REQUEST\_METHOD
- SERVER\_NAME
- SERVER\_PORT
- SERVER\_PROTOCOL
- SERVER\_SOFTWARE

## Requesting extra CGI environment variables

In addition to the CGI environment variables automatically passed to your application code with each query, WebLink applications may request additional variables by creating a comma-separated list of CGI variables in reserved key variable **'MGWCGI'**.

### *For Example:*

```
MGWCGI=ALL_HTTP,CERT_SERIALNUMBER
```

This will result in the additional CGI environment variables **ALL\_HTTP** and **CERT\_SERIALNUMBER** being passed to your application code. Additional CGI environment variables are passed to your Caché application in the %CGIEVAR array, along with the usual list.

## Using Cookies in WebLink

As of Version 2.4, cookies are directly supported within WebLink, as opposed to indirectly via the use of JavaScript. Note that you can still manipulate cookies using JavaScript, if you wish.

Cookies are set by means of the HTTP header directive *SetCookie*, the general form of which is as follows:

**Set-Cookie:** <cookie\_name>=<cookie\_value>; path=/;

This is the minimal specification where *cookie\_name* refers to the name of the cookie, and *cookie\_value* is the associated value. Be

sure to include all other properties in this line. The *path* attribute refers to the path containing the forms with which the cookie is to be returned. The root path specification shown above is adequate for WebLink-generated forms.

An example of setting a cookie would be:

```
Set-Cookie: MyCookie=MyCookieValue; path=/;
```

The cookie name/value pair will be returned (to Caché) in the **HTTP\_COOKIE** CGI environment variable.

More information about Cookies can be obtained by viewing [RFC Number 2109](#)

## Troubleshooting

Most problems encountered in WebLink are due to one of the following:

- Failure to connect to the target Caché system.
- Problems in the Caché application code.

Connection problems are best diagnosed and solved by setting WebLink's on-board diagnostic probes and event logs. This is done by assigning values to the [Env\\_Parameters](#) configuration parameter. All events and errors will be recorded in WebLink's Event Log. Use the information recorded to diagnose and solve your connection problems.

Owing to the inherently stateless nature of the Web, problems in application code are often difficult to track-down. WebLink records information about Caché-side errors in the server-side log. The contents of this log may be viewed by running routine **%mgwe**.

The server-side log may be cleared by running **kill^%mgwe**



## Viewing the Event Log

If you log Caché WebLink information, you can view that information. All entries in the log file are time-stamped. You should clear out your log periodically to keep the file from becoming too large to manage.

To view the event log:

1. Open the Caché WebLink Systems Management page of the Web server on which Caché WebLink is running.
2. Select **View or Clear the Event Log**.
3. To clear the log, click the **Clear Log** button.

For more information on logging, see the description of the Env\_Parameters system parameter.





# Web Event Broker

- [Configuration of the Event Broker](#)
  - [Communicating with Caché](#)
  - [Setting Security for the Applet](#)
  - [Localizing Application Pages](#)
  - [Calling the Applet's Methods in Web Pages](#)
  - [Using the Applet in Static Web Pages](#)
  - [Troubleshooting](#)
  - [Running the Demonstration](#)
- 

WebLink's Java applet allows Caché functionality to be directly invoked as a result of GUI events raised within an HTML (or DHTML or XML) form. WebLink developers can take advantage of a tight coupling between Caché's object model and the various display elements rendered within the client's browser environment and, as such, it is possible for Caché to interact with the document's object model.

## Configuration of the Event Broker

The WebLink Java applet (mgw.class) must reside in a directory accessible to the Web server. The examples in this documentation will be based upon the applet residing in the documents root directory:

- `./wwwroot` for Microsoft Web servers
- `./docs` for Netscape Web servers

The Java applet (which will be running on the client) will communicate, initially with WebLink (running on the Web server). WebLink will forward messages received from the applet to the appropriate Caché server, and return the results subsequently received from the server-side Caché method invoked. The part of WebLink responsible for processing applet communications will be referred to as the **Web Event Broker**, or more simply, the **Event Broker**.

There are two parameters that you must configure for the Event Broker:

- A TCP port must be reserved (on the Web server machine) for handling communications between the applet and WebLink. The configuration parameter [Web\\_Event\\_Broker\\_Port](#) identifies this port. This parameter is maintained via WebLink's [Systems Management Main Menu](#).
- The configuration parameter [Number\\_of\\_Event\\_Brokers](#) specifies the number of individual Event Brokers you wish to start. WebLink supports a maximum of 8 Event Brokers. As a rule of thumb, the greater the number, the greater the performance on busy systems. For configurations where more than one Event Broker is required, separate TCP ports are allocated after, and including, the value specified in the `Web_Event_Broker_Port` parameter.

**Note:** Setting `Number_Of_Event_Brokers` to zero effectively switches off the Event Broker.

The Event Broker will start as soon as a TCP port is filed and the service should appear in WebLink's [System Status](#) screen. Once configured, the Event Broker will automatically restart on future WebLink start-ups.

Note that the examples below are based on a single Event Broker listening on port 7001.

Note also that WebLink's applet is small in size. However, there is a small overhead in downloading the applet to the client (the browser). You should therefore download the applet once, by associating the applet with either a hidden frame or a frame that is permanently on the browser's window. Other frames can still call the applets functions (the [demonstration](#) shows how this is done).

## Communicating with Caché

The first step in using the Event Broker is to include WebLink's Java applet in the HTML form with the following code:

```
<APPLET NAME=WebLink CODEBASE=/ CODE=mgw.class WIDTH=2 HEIGHT=2>
<PARAM NAME=WebEventBrokerPort VALUE=7001>
</APPLET>
```

The *codebase* must be specified and is the directory (relative to the Web server's documents root) where the applet (mgw.class) resides. In the above example, the applet is made available from the document root itself (hence, "CODEBASE=/"). The applet takes a single parameter (**WebEventBrokerPort**) which must be supplied. This parameter specifies the port on which the Event Broker is listening (7001 in this example).

Note that character case is important in parameter names. The value to be assigned to "WebEventBrokerPort" will be supplied to your Caché Web application as the value of the WebLink reserved variable **MGWEBP**. Therefore, the above HTML fragment would normally be generated as follows:

```
write "<APPLET NAME=WebLink CODEBASE=/ CODE=mgw.class WIDTH=2 HEIGHT=2>"
write "<PARAM NAME=WebEventBrokerPort VALUE=", $get(%KEY("MGWEBP")), ">"
write "</APPLET>"
```

"MGWEBP" will be automatically supplied (as part of the incoming form data) for all requests received from WebLink (provided, of course, that at least one Event Broker is running and available for work).

Under normal circumstances, the Java applet automatically determines the IP address of the Web server. However, this functionality is not implemented for all Java engines and access violations can occur when attempting to invoke the applet's functions. If you experience such problems, specify the Web server's IP address explicitly as the value of parameter "WebServerIPAddress":

```
set IPADDR=$get(%CGIEVAR("SERVER_NAME"))
write "<APPLET NAME=WebLink CODEBASE=/ CODE=mgw.class WIDTH=2 HEIGHT=2>"
write "<PARAM NAME=EventBrokerPort VALUE=", $get(%KEY("MGWEBP")), ">"
write "<PARAM NAME=WebServerIPAddress VALUE=", IPADDR, ">", CRLF
write "</APPLET>"
```

Note that minimal values are provided for the "HEIGHT" and "WIDTH" parameters. This is advisable because some browsers may not run an applet if no screen space is reserved. These minimal values should be no less than 2 units.

## Setting Security for the Applet

WebLink's applet may communicate with Caché by one of three methods. The **Mode** parameter indicates which communications protocol to use:

Mode	Description
0	Standard mode of operation through the Event Broker (the default if the Mode parameter is not supplied). This mode offers the best performance.
10	Uses the browser's "http" stream as the transport. In terms of security, this is equivalent to Mode 0. However, this mode can operate independently of WebLink's Event Broker service (i.e., the Event Broker need not be running). Expect this mode of operation to be slightly slower than Mode 0.
11	Attempts to use the browser's "https" stream as the transport. This is the high security option, based on SSL.

To use the http-based modes (i.e. 10 and 11), the **ScriptName** parameter must be set when the applet is loaded. This should take the value of the "script name" held in WebLink's reserved variable **MGWLIB**. For example:

```
write "<PARAM NAME=Mode VALUE=10>"
write "<PARAM NAME=ScriptName VALUE=", $G(%KEY("MGWLIB")), ">"
```

Note that the value of **MGWLIB** will be equivalent to the CGI environment variable **SCRIPT\_NAME**. This will be **/scripts/mgwms32.dll** for most WebLink applications.

The mode of communication may be dynamically changed from within the client environment (i.e., through JavaScript) by calling the **ChangeMode** function:

```
success = ChangeMode(String NewMode);
```

For example:

```
success = ChangeMode("11");
```

The value of **success** will be returned as "1" if the mode was successfully changed or "0" otherwise. You cannot use this function to swap out of the secure mode (i.e., Mode=11).

The mode under which the request was sent will be recorded in Caché variable **%REQUEST(0)** for the benefit of your application code.

## Localizing Application Pages

The applet supports multi-byte character sets. In general, the applet will attempt to derive the character-encoding scheme from the hosting environment (i.e., the Web browser).

However, you can control the encoding scheme from your application by specifying the scheme in the **CharacterEncoding** parameter. For example:

```
<PARAM NAME=CharacterEncoding VALUE="8859_1">
```

The default encoding scheme in WebLink is ISO Latin-1 (8859\_1). This default will be used when the application does not specify a scheme and auto-detection is not possible.

Encoding schemes supported by WebLink (as defined in Java version 1.1) are listed below:

CharacterEncoding	Description
8859_1	ISO Latin-1 (English Text)
8859_2	ISO Latin-2
8859_3	ISO Latin-3
8859_4	ISO Latin-4
8859_5	ISO Latin/Cyrillic
8859_6	ISO Latin/Arabic
8859_7	ISO Latin/Greek
8859_8	ISO Latin/Hebrew
8859_9	ISO Latin-5
Big5	Big 5 Traditional Chinese
CNS11643	CNS 11643 Traditional Chinese
Cp1250	Windows Eastern Europe / Latin-2
Cp1251	Windows Cyrillic
Cp1252	Windows Western Europe / Latin-1
Cp1253	Windows Greek
Cp1254	Windows Turkish
Cp1255	Windows Hebrew

Cp1256	Windows Arabic
Cp1257	Windows Baltic
Cp1258	Windows Vietnamese
Cp437	PC Original
Cp737	PC Greek
Cp775	PC Baltic
Cp850	PC Latin-1
Cp852	PC Latin-2
Cp855	PC Cyrillic
Cp857	PC Turkish
Cp860	PC Portugese
Cp861	PC Icelandic
Cp862	PC Hebrew
Cp863	PC Canadian French
Cp864	PC Arabic
Cp865	PC Nordic
Cp866	PC Russian
Cp869	PC Modern Greek
Cp874	Windows Thai
EUCJIS	Japanese EUC
GB2312	GP2312-80 Simplified Chinese
JIS	JIS
KSC5601	KSC5601 Korean
MacArabic	Macintosh Arabic
MacCentralEurope	Macintosh Latin-2
MacCroatian	Macintosh Croatian
MacCyrillic	Macintosh Cyrillic
MacDingbat	Macintosh Dingbat
MacGreek	Macintosh Greek
MacHebrew	Macintosh Hebrew
MacIceland	Macintosh Iceland
MacRoman	Macintosh Roman
MacRomania	Macintosh Romania
MacSymbol	Macintosh Symbol
MacThai	Macintosh Thai
MacTurkish	Macintosh Turkish
MacUkraine	Macintosh Ukraine
SJIS	PC and Windows Japanese
UTF8	Standard UTF-8

## Calling the Applet's Methods in Web Pages

Functionality within Caché is exposed to the client's (i.e., browser) environment by means of the [CacheMethod](#) and [CacheMethod\\_Multiple](#) functions. These functions are usually called via short JavaScript modules contained within the HTML form.

### CacheMethod Function

The CacheMethod function returns a single value from Caché. The function syntax is:

CacheMethod(MGWCHD, DataObject, Parameters)

where:

MGWCHD	The WebLink connection handle supplied to your Web application by WebLink. This will identify either your Caché system for state-less applications or your private Caché process for state-aware applications. Simply include this parameter as you receive it (i.e., %KEY("MGWCHD")). For clarity, this parameter will be shown as 0 (zero) in the examples that follow.
DataObject	The name of the Caché operation (method) to be invoked on the server. This can be any name of your choosing. However, for Caché systems it is recommended that this parameter takes the full name of the Caché method. For Open M systems, this parameter could take the name of an M routine/procedure or could be plain text. Your Web application will ultimately be responsible for processing this value and invoking the appropriate code.
Parameters	The list of inputs expected by the operation specified in 'DataObject'. Multiple input parameters are separated by the ASCII 1 character. Alternatively, the parameter list may be packaged using an <b>Aggregate_n</b> function where <i>n</i> is the number of input parameters. <b>Aggregate_n</b> takes <i>n</i> arguments - the list of input values (in order).

The following two calls are equivalent:

```
result = CacheMethod("0", "MyClass.MyMethod", "input 1" + "\001" + "input 2")
```

```
result = CacheMethod("0", "MyClass.MyMethod", Aggregate_2("input 1", "input 2"))
```

### CacheMethod Example

Consider a form where a company's customer details are maintained. The first data entry point on this form is the customer's account number. As soon as this field is entered, we wish to invoke a Caché method to verify that the number is already held on file. The HTML generated by WebLink for such a data-entry form might look like this:

```
<HTML>
<HEAD><TITLE>Customer Contacts</TITLE></HEAD>
<BODY>
<FORM>
<H2>Customer Contacts</H2>
<HR>
<BR>Customer Number:
<INPUT TYPE=TEXT NAME=CustNo SIZE=25 ONCHANGE="ValidateCustNo(this.value)">
<BR>Contact Office:
<INPUT TYPE=TEXT NAME=ContactOffice SIZE=25>
<BR>Contact Name:
<INPUT TYPE=TEXT NAME=ContactName SIZE=25>
<BR><INPUT TYPE=BUTTON VALUE="File Data">
<HR>

<SCRIPT LANGUAGE = "JavaScript">
function ValidateCustNo(Value) {
  error = document.WebLink.CacheMethod("0", "Customer.CustNo.Validate", Value);
  if (error != "") {
    alert(error);
    return false;
  }
  else {
    return true;
  }
}
```

```

</SCRIPT>

<APPLET NAME=WebLink CODEBASE=/ CODE=mgw.class WIDTH=2 HEIGHT=2>
<PARAM NAME=EventBrokerPort VALUE=7001>
</APPLET>

</FORM>
</HTML>

```

In this form, the JavaScript function "ValidateCustNo" is invoked when the customer number field (CustNo) is changed. This JavaScript function is responsible for sending the user input (CustNo.value) to Caché together with the Caché method name (Customer.CustNo.Validate). Essentially, Caché will evaluate:

```
Customer.CustNo.Validate(CustNo.Value)
```

This particular method is designed to return an error if there is something wrong with the customer number entered, or null if the number is valid. The JavaScript function will display the error (if any) in an alert dialogue box.

The corresponding Caché method (implemented as ObjectScript here) might look like this:

```

CCV ; Customer.CustNo.Validate
    set CustNo=$get(%REQUEST(1))
    if CustNo="" do sendline^%mgwj("A Customer Number must be entered") quit
    if '$defined(^CUST(CustNo)) do sendline^%mgwj("No Customer is registered under this
number") quit
    do sendline^%mgwj("") ; No error
    quit

```

Notice the use of the **SENDLINE** function to send data back to the client. This function should always be used because it automatically handles the data packaging required by WebLink.

## Launching Your Caché Application Code to Process the User's Query

The mechanisms through which this Caché procedure is invoked on the server will depend upon whether the Web application is:

1. stateless, or state-aware **and** not using the SEND method.
2. state-aware **and** using WebLink's SEND method.

**1:** For stateless applications (or state-aware applications **not** using the SEND method), the calling mechanism is similar to that used for the processing of whole forms. The appropriate Caché code is launched from the JAVA section of the same routine from which your form-processing code is launched (%MGW2 or %ZMGW2):

```

JAVA ; Process in-form Caché Method (invoked by Java applet)
    if %REQUEST="Customer.CustNo.Validate" do CCV^MyApp quit
    do TESTJ
    quit

```

As an alternative to the routine-based launch method described above, application code can also be recorded in the ^MGWMETH global.

For our example, the method **Customer.CustNo.Validate** will translate to either:

```
^MGWMETH("Customer.CustNo.Validate")="CCV^MyApp"
```

or

```
^MGWMETH("Customer","CustNo","Validate")="CCV^MyApp"
```

WebLink will look for a Caché method in both records. If WebLink fails to find a suitable method or procedure in global **^MGWMETH**, it will transfer control to the launch routine - %MGW2 (or %ZMGW2).

**2:** If your application is state-aware *and* using the SEND method, this request would be processed as follows:

```
        ; HTML form complete, wait for user action
FORM1 do send^%mgw
        ; Process requests received from Java applet
        if $get(%REQUEST)="Customer.CustNo.Validate" do CCV^MyApp goto FORM1
        ; Process completed form
        ;
```

Note that the Caché variable **%REQUEST** is only defined for requests originating from the applet functions (hence the use of the \$get function).

## CacheMethod\_Multiple Function

Use the CacheMethod\_Multiple function to return a set of values (a list) from Caché. The function syntax is:

CacheMethod\_Multiple(*MGWCHD, DataObject, Parameters*)

The function's parameters are the same as for the [CacheMethod](#) function. The function actually returns True or False (False at the end of a record set).

The CacheMethod\_Multiple function in turn uses the CacheMethod\_GetComponent to return a specific data component from a record. The syntax of this function is:

CacheMethod\_GetComponent(*MGWCHD, DataComponentName*)

Rows of data are returned by the CacheMethod\_Multiple function. This function should be processed in a JavaScript WHILE loop until the result is "False" (end of data set). On each pass, individual data components can be extracted from the record using the CacheMethod\_GetComponent function.

Say, for example, you have a method (**Customer.List**) that generates a list of all your customers. For each customer record (or instance of a customer), the customer's account number (Customer.CustNo) and the customer's name (Customer.Name) are returned. The scheme for implementing this list through JavaScript is as follows:

```
while (WebLink.CacheMethod_Multiple("0", "Customer.List", Parameters)) {
    /* Get specific components */
    CustNo = WebLink.CacheMethod_GetComponent("0" "Customer.CustNo");
    Name = WebLink.CacheMethod_GetComponent("0" "Customer.Name");
    /* Process list member */
}
```

The corresponding Caché method must, for the benefit of the CacheMethod\_GetComponent function, send the names of the individual data components as well as the data values themselves. By convention, this information is sent to the client as the first data record by using the SENDHEAD function. The columns for both headings and data values are entered in Caché array %TXT. The Caché method for the above example is shown below (implemented as an ObjectScript function):

```
CL ; Customer.List
set %TXT(1)="Customer.CustNo"
set %TXT(2)="Customer.Name"
do sendhead^%mgwj("") ; Send column headings
set CustNo=""
for set CustNo=$Order(^CUSTOMER(CustNo)) quit:CustNo="" do CL1
do sendline^%mgwj("") ; No error
quit
;
CL1 ; Retrieve the customer's record and send to the client
```

```

set Name=$piece($get(^CUSTOMER(CustNo)), "#", 1)
set %TXT(1)=CustNo,%TXT(2)=Name
do sendline^%mgwj("") ; Send data record
quit

```

It is possible to use the CacheMethod\_GetComponent function with the "single record" case (as returned by the CacheMethod function) provided, of course, that the Caché method has sent the component names (or record header) to the client.

For example, consider the case where a method returns a single customer record. Customers are uniquely identified by their account numbers. Therefore, the Caché method for returning a single record could be specified as:

```
Customer.GetRecord(Customer.CustNo)
```

The JavaScript call to this Caché method:

```

Record = CacheMethod("0", "Customer.GetRecord", CustNo));
/* Get specific components */
Name = WebLink.CacheMethod_GetComponent("0", "Customer.Name");
Address = WebLink.CacheMethod_GetComponent("0", "Customer.Address");

```

The Caché Method:

```

CGR ; Customer.GetRecord
set CustNo=$get(%REQUEST(1))
set %TXT(1)="Customer.Name"
set %TXT(2)="Customer.Address"
do sendhead^%mgwj("")
Record=$get(^CUSTOMER(CustNo))
set Name=$piece(record, "#", 1)
set Address=$piece(record, "#", 2)
set %TXT(1)="Name"
set %TXT(2)="Address"
do sendhead^%mgwj("")
quit

```

## A Further Example

Returning to the HTML form listed previously, let's extend the functionality to allow the customer's contact office (ContactOffice) to be selected from a list. The list of offices is held in the Caché database and is too large to be routinely downloaded with the form. We will allow the user to request a list of offices at the point of entry. We will also allow the user to restrict the size of the list by specifying part of an office name.

The HTML and JavaScript implementation of this functionality is shown below. The same JavaScript could potentially be used for processing many list objects on the form -- the data object and method names are passed as arguments to the "Select" function.

```

Contact Name:
<INPUT TYPE=TEXT NAME=ContactName SIZE=25>
<BR><INPUT TYPE=BUTTON VALUE="Show List"
ONCLICK="Select(form.ContactOffice,
    'ContactOffice.List',
    'ContactOffice.Name',
    ContactOffice.value)">

<SCRIPT LANGUAGE = "JavaScript">
function Select(FormObject, ListObject, DataObject, SearchSeed) {
while (WebLink.CacheMethod_Multiple("0", ListObject, SearchSeed)) {
    /* Get specific component for display */
    Name = WebLink.CacheMethod_GetComponent("0" DataObject));
    /* Add Name to the forms list object */
}
}
</SCRIPT>

```



The corresponding Caché Method:

```
CNL ; ContactOffice.List
    set SearchSeed=$get(%REQUEST(1))
    set %TXT(1)="ContactOffice.Code"
    set %TXT(2)="ContactOffice.Name"
    do sendhead^%mgwj("")
    set Code="" for set Code=$Order(^CITY(Code)) quit:Code="" do CNL1
    quit
;
CNL1 ; Check the office record and send to the client
    set Name=$get(^CITY(Code))
    if Name'[SearchSeed quit ; don't send
    set %TXT(1)=Code,%TXT(2)=Name
    do sendline^%mgwj("")
    quit
```

## Using the Applet in Static Web Pages

The applet's functionality can be used in static Web pages (i.e., Web pages not served by WebLink).

To do this, set the value of the connection handle (MGWCHD) to the name of the target Caché system (as defined within WebLink).

JavaScript calls like the one shown in the following example can be made within static pages.

```
error = WebLink.CacheMethod("MyCacheServer", "Customer.CustNo.Validate", Value);
```

This facility provides an easy mechanism for introducing Caché functionality into an existing static web site. Only state-less communication with Caché is supported in static pages.

The Event Broker must be active for this facility to work in Communications [Mode 0](#) (the default). To start the Event Broker, use WebLink's [Systems Management Main Menu](#). This will effectively force the Web server to load the WebLink library - the Event Broker will automatically start at this point. The Event Broker need not be active if you have instructed the applet to communicate with Caché via the browser's 'http' streams (i.e., [Mode 10](#) or [Mode 11](#)).

## Troubleshooting

Include **e** in the global [Env\\_Parameters](#) parameter to cause all errors resulting from Event Broker communications to be recorded in the event log.

Include **j** in the global Env\_Parameters to record all Event Broker transactions in the event log. It is recommended that you use **j** only during development or troubleshooting, because it generate a large volume of data in the event log. Using **j** will record all four aspects of the networking protocol:

- Requests successfully received by WebLink from the Java Applet.
- Requests successfully sent by WebLink to the Caché system.
- Data successfully received by WebLink from the Caché system.
- Data successfully sent by WebLink to the Java Applet.

To summarize, the global directive:

```
Env_Parameters=ej
```

will switch on all diagnostic probes relevant to the Event Broker.

## Running the Demonstration

A demonstration of the technology is included in the WebLink distribution files. This demonstration shows an extended version of the application described above.

The demonstration is contained within one routine (**%MGWJDEM**) which may be imported from **mgwjdem.rou**. All the data required to run this application is hard-coded into this routine. The demo form is designed to run as part of a stateless Web application. After loading this routine, modify your copy of %MGW2 (or %ZMGW2) to include the two lines shown in boldface italics:

```
HTML ; Process incoming HTML data
    if $defined(%KEY("mgwj")) do ^%MGWJDEM quit
do TEST
quit
;
JAVA ; Process in-form Cache Method (Invoked by Java applet)
; Receive request in %REQUEST
; Associated parameters in %REQUEST(1->n)
; Send each line of response as sendline^%mgwj(<Data>)
; Finally, quit from this code module
;
if %REQUEST?1"Customer".E!(%REQUEST?1"Contact".E) do JAVA^%MGWJDEM quit
do TESTJ
quit
;
```

You can now run the application. Submit the following URL from your browser:

```
http://127.0.0.1/scripts/mgwms32.dll?MGWLPN=<C_Server>&mgwj=0
```

where *C\_Server* is the name of your Caché server.

The application only contains one HTML form. The "File Data" button does not do anything. Customer Number is validated by a Caché method. An error dialogue is shown if an invalid number is entered. If a valid number is entered, a second Caché method is invoked to retrieve the customer's name, which is displayed in the second text entry box. This part illustrates how Caché functions can be called from HTML and how the results can be used to modify the form's Document Object Model. Only customer number "100000" is recognized in the demonstration (this corresponds to a customer name of "InterSystems Corporation").

The "Contact Office" field has an associated button for showing a list from which you may select an office name. The list appears in the second frame. Users may also enter a synonym for the office name (instead of requesting a list). For example, try entering "o" (for Oxford) or "sf" (for San Francisco). The Synonym is automatically expanded into the corresponding full name (if any is found) by a Caché method (another example of Caché functionality manipulating the Document's Object Model).



# PDQweb Compatibility

- [Compatibility between Caché WebLink and PDQweb](#)
  - [Running PDQweb Applications with Microsoft Web Servers](#)
  - [PDQweb's Application Files](#)
  - [The %MSMWEB Routine](#)
  - [Running PDQweb Applications with Netscape Web Servers](#)
- 

## Compatibility between Caché WebLink and PDQweb

PDQweb is the Web connectivity product developed by Micronetix for their Micronetix Standard M (MSM) product. Micronetix' product line is now owned by InterSystems and Caché WebLink has been adapted to run Web applications developed using PDQweb.

Caché WebLink will run PDQweb applications. You do not need to make any changes to your PDQweb application code.

**Note:** This document refers to Caché systems throughout - WebLink will, in fact, work with any Open M product including, of course, MSM.

## Running PDQweb Applications with Microsoft Web Servers

The instructions given here assume that you already have PDQweb installed with your Microsoft Web server installation.

The PDQweb directory tree (**/PDQweb**) lives under the documents root directory (i.e., **/inetpub/wwwroot/PDQweb**).

### Modifying PDQweb's Registry Setting

PDQweb's running configuration includes a Registry setting which instructs the Web server to pass control to the PDQweb library when a PDQweb application file is requested by a client. PDQweb's application files are identified by the **.web** file extension.

PDQweb's Registry setting must be modified such that **.web** files are processed by WebLink instead of PDQweb.

The registry entry for PDQweb is:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Script Map]
".web"="C:\Inetpub\wwwroot\PDQweb\bin\PDQweb.dll"
```

This should be modified to:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Script Map]
".web"="C:\Inetpub\scripts\mgwms32.dll"
```

Assuming that you have installed WebLink on your machine as [documented in this guide](#), this is the only required change. Caché WebLink will now run your PDQweb applications. However, it is recommended that PDQweb's applications files are modified as suggested in the following section.

## PDQweb's Application Files

Information about which Caché system to connect to (i.e., the equivalent of WebLink's **mgw.ini** file) is held in each of PDQweb's application files (i.e., the **.web** files that you request in your URLs).

These files contain the following four sections:

## MAPPINGS

This section contains the list of mappings which relate the value of the **'EP'** variable in your request to the corresponding Caché-call in your PDQweb application. WebLink will read and interpret this section. No action is required here.

## DATASOURCES

This section identifies the target Caché (or MSM) system. WebLink will scan this section and choose the most appropriate Caché system from its own configuration. A match is made on the basis of IP address. You can specify the Caché (or MSM) system you wish WebLink to use by specifying a value for the **MGWLPN** parameter in this section.

For example, an existing PDQweb configuration for a local system might be:

```
[DATASOURCES]
127.0.0.1:2001:PASSWORD:3:30
```

To instruct WebLink to use an Caché system of your choice (say **'MY\_LOCAL\_MSM'**), modify this section to read:

```
[DATASOURCES]
MGWLPN=MY_LOCAL_MSM
127.0.0.1:2001:PASSWORD:3:30
```

Of course, **'MY\_LOCAL\_MSM'** must exist as a properly configured Caché system within WebLink's configuration.

It is strongly recommended that you use this facility to formally specify the Caché system to use - this will resolve any conflicts caused by many Caché configurations sharing the same IP address within WebLink.

## CGI

This section specifies the CGI environment variables to be sent to Caché. WebLink will process this section and send any CGI environment variable not already included in its list (WebLink always sends the common CGI environment variables with each and every request).

## DEBUG

Micronetics' 'Debug' facility - WebLink ignores this section.

## The %MSMWEB Routine

PDQweb applications call back to this routine to set Cookies (**COOKIE^%MSMWEB**) and to perform URL redirections (**REDURL^%MSMWEB**). If you have an existing PDQweb installation, WebLink will work with Micronetics' version of this routine. If not, install the 'WebLink' version from **msmweb.rou**.

However, it is recommended that you handle Cookies and URL redirections by formulating proper HTTP headings where required.

## Running PDQweb Applications with Netscape Web Servers

You can use WebLink to run your PDQweb application under the Netscape Web servers (NT and UNIX). Assuming that WebLink is already installed, proceed as follows:

1. Install the PDQweb directory tree under Netscape's **/docs** directory (documents root).
2. Modify the **text/html** line in the **mime.types** file to read:

```
type=text/html exts=htm,html,dll,web
```

3. Add the following section to the end of the **obj.conf** file:

```
<Object ppath="*/PDQweb/bin/*">  
Service method=(GET|HEAD|POST) fn=CacheWebLink  
</Object>
```

4. Restart the Netscape Web server.

5. Run your PDQweb application.

If you port a PDQweb application to a UNIX-based Netscape Web server, one thing to remember is that UNIX is case-sensitive. i.e. the **/PDQweb/bin** specified in your URL must be the same case as the **/PDQweb/bin** specified in your **obj.conf** which must be the same case as the **/PDQweb/bin** directory set up under the documents root (**/docs**).

You will find that the hyperlinks in some of Micronetics' examples will need modifying for this reason.



# Installation

- [Supported Platforms for Caché WebLink](#)
  - [Preparing Your Systems for Caché WebLink Installation](#)
  - [Installing Caché WebLink](#)
- 

This chapter describes the requirements for installing WebLink, as well as the installation procedure itself.

## Supported Platforms for Caché WebLink

Caché WebLink is supported on the following InterSystems database servers:

- Caché 3.1
- Caché 2.1.x
- DSM 6.6 and 7.x (for OpenVMS platforms)
- ISM 6.2 or later (for UNIX or Alpha OpenVMS platforms)
- MSM 4.x

These databases must:

- Have an Internet or intranet connection with TCP support.
- Allows TCP access.

The Web server must be one of the following:

- Microsoft IIS 1.0 and later (Windows NT)
  - Microsoft Personal Web Server (Windows)
  - Microsoft Peer Web Server (Windows)
  - Netscape 2.x and greater Web servers (Windows and UNIX)
  - SCO FastStart 2, 3
- 
- Apache - refer to the [Apache Web Servers](#) section.
  - Any web server offering a CGI interface on a supported operating system - refer to the [WebLink Network Service Daemon](#) section.

Clients must use a Web browser such as Netscape Navigator (3.x or later), Netscape Communicator, or Microsoft Internet Explorer 4.0 or later.

## Preparing Your Systems for Caché WebLink Installation

To prepare your Caché (or Open M) database server and your Web server for Caché WebLink installation:

1. If your database server runs on an **OpenVMS** platform, you must configure terminal sessions for 8-bit raw mode so that character interrupts and backspaces are not processed by the telnet client and foreign character sets display properly. The following is an example of a DCL command that enables 8-bit raw mode:

```
$ setterm/perm/eight/pasthru/host/nowrap
```

Note that this command may be different for different versions of OpenVMS. Consult your system documentation for the exact command.

2. Caché for Windows 95/98 systems and any system using 2-byte character sets require using socket connections. For all other connections you can use either socket or TELNET connections.

3. If your Web server runs on a **UNIX** system, you must configure your httpd to use only a single process for each Web request and allow as many threads as there are Caché WebLink sessions. To do this on a Netscape Web server:
  - a. Open the **magnus.conf** file in the Web server directory.
  - b. Set the **MaxProcs** parameter to 1.
  - c. Set the **MaxThreads** parameter to at least the value set in Caché WebLink's [Max\\_Sessions](#) parameter.
4. Start Caché on the servers you want to configure for use with Caché WebLink.
5. To test telnet availability, connect to the desired Caché systems using a telnet client from the machine hosting your Web server. If you cannot reach a system, do not proceed until you fix the connection.
6. Shut down all Internet services on the machine hosting the Web server. These include gopher servers, ftp servers, Web servers, and news servers.

## Installing Caché WebLink

The Caché WebLink installation procedure consists of installing these components on either the Web server or the Caché database server:

1. Caché WebLink **libraries** on the Web server. These libraries (DLLs for Windows or shared object libraries for UNIX and OpenVMS) provide a link between the Web server and the WebLink routines on the Caché database server.
2. Caché WebLink [Java applet](#) on the Web server. The applet will communicate between the user's browser and the Web Event Broker.
3. Caché WebLink **routines** on the Caché (or Open M) database server.
4. Caché WebLink [Network Service Daemon](#) on the Web server and on the Caché or Open M database server. This optional component is used for applications that use CGI and for Web servers that do not support ISAPI and NSAPI.
5. Caché WebLink [Developer](#) on the Caché (or Open M) database server. This utility lets you design an application as a set of Web pages with embedded Caché ObjectScript code.

It is assumed that you either have a Caché 3.1 for Windows production CD or have downloaded Caché WebLink from the [InterSystems WebLink Web page](#).

## Caché for Windows Installation

Your Caché 3.1 for Windows product distribution CD contains Caché WebLink. Use your *Caché for Windows Installation Guide* to perform a Standard or Custom installation of Caché 3.1 on the Windows server. The installation program will:

- Load the Caché WebLink routines (both core routines and WebLink Developer routines) onto the Caché server. The default location for the WebLink software is **CacheSys\WebLink**.
- Install the Caché WebLink DLLs on the Web server (if the Web server and the Caché server are on the same machine). For example, on an NT Personal Web Server, the **mgwms32.dll** file is installed in the **Inetpub\scripts** directory.
- Install the Caché WebLink Java applet (if the Web server and the Caché server are on the same machine). For example, on an NT Personal Web Server, the **mgw.class** file (containing the Java applet) is installed in the **Inetpub\wwwroot** directory.

If your Web server runs on a different machine than your Caché database, you need to install the Caché WebLink libraries on the machine hosting your Web server:

- For Microsoft Web servers, copy **mgwms32.dll** to the Web server's **scripts** directory (usually InetPub/scripts).

- For Netscape Web servers, copy **mgwns20.dll** or **mgwns30.dll** to the Web server's **scripts** directory (usually `.../httpd-servname/scripts/` or `.../https-servname/scripts/`). Use **mgwns20.dll** for Netscape API version 2.x (including all versions of FastTrack) and **mgwns30.dll** for Netscape API version 3.x. Note that you will have to create the scripts directory on new Netscape installations.

If you intend to use the Caché WebLink [Network Service Daemon](#):

1. Copy **mgwmt.exe** to the **CacheSys\WebLink** directory.
2. If you are using WebLink's CGI modules, copy the CGI executables to the Web server's CGI directory. The executables are:
  - **mgwcgi** and **nph-mgwcgi** for UNIX-based systems
  - **mgwcgi.exe** and **nph-mgwcgi.exe** for NT-based systems

After you have installed Caché WebLink, proceed to [Express Configuration](#) for a quick guide to configuring your system.

## Installing from the World Wide Web

To install the Caché WebLink download version available on the InterSystems [WebLink Web page](#):

1. Download the Caché WebLink self-extracting archive (for Windows 95/98 and NT) or the tar file (for UNIX) from the [WebLink Web page](#).

**Note:** InterSystems recommends saving the UNIX archive in an empty temporary directory.

2. On a **Windows** system, double-click on the .exe file you downloaded. This begins the install process. On a **UNIX** system untar the file and run `./Setup`.
3. During setup you can specify which components to install and where to install them. This program contains all of the necessary components for Caché WebLink. The default location is **\CacheSys\WebLink** on the same drive as your Windows directory or, if you have Caché installed, the same drive as Caché. On UNIX, the default location is **/usr/cachesys**. The routines are stored in the **routines** subdirectory of the user-specified directory.

**Note:** The organization entered into Caché WebLink becomes the default site name in your Caché WebLink setup. This name must match the site name in your license, if you have one.

**Caution:** For Netscape Web servers, Caché WebLink requires modifications to your Netscape configuration files. The installation program displays the lines it needs to add and which configuration files contains those lines. Caché WebLink does this automatically. If these lines conflict with your current configuration, you should manually [customize](#) your Netscape Web server configuration files.

4. After the installation program finishes, restart your Web server. It is also safe to restart any other Internet services running on the machine hosting your Web server.
5. Load the Caché WebLink routines on the target servers. The routines are stored in **mgw.rou**. The routine names are **%mgw**, **%mgwO**, **%mgwa**, **%mgwb**, **%mgwd**, **%mgwe**, and **%mgwj**.

**Note1:** For Caché version 3.x, you should install all **%mgw\*** routines in the **%CACHELIB** NameSpace.

**Note2:** For all Caché systems you should load routine **%mgw1** from **mgw1.rou**. This routine is distributed in object form - use routine **%RIMF** to load it.



**Note3:** The **OpenVMS** save routine mgw.rou can be downloaded directly to your OpenVMS system in ASCII or transferred from your UNIX or Windows system.

**Note4:** In previous versions of Caché WebLink, the %MGW2 routine came included in the package. If you have an existing %MGW2 (or %ZMGW2), Caché WebLink will continue to use that file as the default for queries, otherwise it will use %mgw3 (the test form). When you build your Caché WebLink applications, place the calls to your applications in %MGW2 (or %ZMGW2). If you only have %mgw3, copy %mgw3 to %ZMGW2 and then embed the calls to your Web enabled applications in the %ZMGW2 routine. Note that users of Caché must call their application code from %ZMGW2 not %MGW2.

Keep in mind that Windows 95 Caché systems and any system using 2-byte character sets require the use socket connections. For all other connections you can use either socket or TELNET connections.

Proceed to [Express Configuration](#) for a quick guide to configuring your system.



# Running Caché WebLink with Apache Web Servers

- [Apache Version 1.2.x and 1.3.x for UNIX and NT Systems](#)
  - [Adding the WebLink Module to Apache](#)
  - [Adding WebLink's CGI Module to Apache](#)
  - [Installing and Operating WebLink's Network Service Daemon](#)
  - [Calling WebLink through Apache](#)
- 

## Apache Version 1.2.x and 1.3.x for UNIX and NT Systems

The Microsoft and Netscape line of Web servers both support a multi-threaded API which allows extensions, in the form of dynamically bound libraries, to be made to the Web server's core functionality. Current versions of WebLink make full use of these APIs in order to bring high-performance Web connectivity to the Caché system.

Apache is different to the Microsoft and Netscape Web servers in that it does not support the Microsoft/Netscape style of API. However, extra functionality can be added to Apache by means of user-defined modules (compiled C programs). In fact, a large part of Apache's core functionality is implemented as a set of modules.

Unfortunately, adding a new module to Apache involves reconfiguring and rebuilding the Web server. A **WebLink module** is provided for this purpose. For those who do not wish to rebuild their Web server, WebLink also provides a high-performance CGI-based solution. Both the WebLink module and the CGI executables are small functional blocks designed to exclusively communicate with WebLink's **Network Service Daemon (NSD)**. The NSD is responsible for providing WebLink's core functionality and persistent links to Caché.

The key decision to make at this point is whether or not you wish to rebuild the Apache Web server in order to add the WebLink module or opt for the less complicated CGI solution instead. The performance of the two approaches appears to be similar though, technically, the bound-in module ought to be faster. Read on to the section describing how you [add WebLink's module](#) for guidance on how to build the module into the Apache core or skip to the section describing how you [add WebLink's CGI module](#) for the CGI-based approach.

## Adding the WebLink Module to Apache

Be sure to read these instructions in conjunction with the specific documentation contained within your Apache distribution. Note that the instructions given here assume that the root directory for the Apache installation is **apache**. In practice, this directory name will usually have the apache version number appended to it. The instructions given here refer to UNIX systems - the procedure for NT/95/98 systems is similar.

### Adding the WebLink Module Source Code to the Apache Source Directory

The code for user-defined modules resides under directory:

```
/apache/src/modules
```

Create the following directory for the WebLink module:

```
/apache/src/modules/weblink
```

While in the **/apache/src/modules** directory, type

```
mkdir weblink
```

Copy the source for the WebLink module (**mod\_weblink.c**) into this directory. You should now have:

```
/apache/src/modules/weblink/mod_weblink.c
```

## Creating a Makefile for the WebLink Module

Now we need to create a **Makefile**. The Makefile contains instructions on how the WebLink module should be compiled and linked. Apache always provides an example module - we will make a copy of the example's Makefile and modify it for WebLink.

Make sure you are in the **weblink** directory:

```
/apache/src/modules/weblink
```

Copy the example Makefile by typing:

```
cp ../example/Makefile* ./
```

Depending on the version of Apache you have, you will find one of the following files in the **weblink** directory:

**For Apache 1.3.x:** Makefile

**For Apache 1.3.x:** Makefile.tpl (Makefile template)

Edit whichever file you have and replace all references to **mod\_example** with **mod\_weblink**.

## Configuring the Apache Build Process to Recognize the WebLink Module

Return to the **src** directory:

```
/apache/src
```

Edit the file: **Configuration**.

**For Apache 1.2.x:**

Near the end of this file you should find the line:

```
# Module example_module modules/example/mod_example.o
```

After this line, add the following line:

```
Module weblink_module modules/weblink/mod_weblink.o
```

**For Apache 1.3.x:**

Near the end of this file you should find the line:

```
# AddModule modules/example/mod_example.o
```

After this line, add the following line:

```
AddModule modules/weblink/mod_weblink.o
```

## Running the Configure Script to Build the Main Makefile

You can now configure the build process. Make sure you are in the **src** directory:

```
/apache/src
```

Type:

```
./Configure
```

## Building Apache on Red Hat Linux

Red Hat Linux is distributed with a pre-built version of Apache. Provided you chose to install Apache as part of the Linux installation procedure, you will find that Apache is automatically started when Linux is booted. The layout of the run-time file system for this pre-built version is slightly different to that which is prescribed as the default layout in the Apache source code. In order to account for this, the following change should be made to the header file **httpd.h**:

*For Apache 1.2.x:*

```
/apache/src/httpd.h
```

*For Apache 1.3.x:*

```
/apache/src/include/httpd.h
```

The Web server root (**HTTPD\_ROOT**) is one of the first items defined in this file. For Red Hat Linux this should be defined as follows:

```
#define HTTPD_ROOT "/etc/httpd"
```

## Building the Apache Executable

We can now build the Apache executable. Make sure you are in the **src** directory:

```
/apache/src
```

Type:

```
make
```

This should produce a new Web server executable (**httpd**) with the WebLink module's functionality built-in. The httpd executable will be created in the **src** directory. For normal operation this should be copied to (and run in) the **apache** directory.

For Red Hat Linux, the executable **httpd** should be copied to:

```
/usr/sbin
```

Note: Be sure to backup the existing copy first.

## Run-time Configuration

Edit the Apache configuration file **httpd.conf**. For the standard Apache distribution this file will be found in:

```
/apache/conf
```

For Red Hat Linux, the run-time version of **httpd.conf** will be found in:

```
/etc/httpd/conf
```

Assuming that you wish to invoke WebLink from the usual **/scripts** location, add the following section to the end of this file:

```
<Location /scripts>  
SetHandler weblink-handler  
</Location>
```

It is not necessary to physically create the **scripts** directory, but it is recommend that you do just in case a previously invoked module (i.e., invoked before WebLink) takes exception to the fact that the **/scripts** directory does not exist. Note that all requests to Apache are serviced by a set of modules invoked in sequence - WebLink is one of these modules, but will be one of the first invoked provided you added its definition near the end of the **Configuration** file as suggested previously.

The **/scripts** directory should be created below the documents root. For the standard Apache distribution this will be:

```
/apache/htdocs
```

Create:

```
/apache/htdocs/scripts
```

For Red Hat Linux, the document's root is:

```
/home/httpd/html
```

Create:

```
/home/httpd/html/scripts
```

Now, proceed to the section describing how to [install and set up WebLink's Network Service Daemon](#).

## Adding WebLink's CGI Module to Apache

Copy the two WebLink-CGI executables (mgwcgi and nph-mgwcgi) into the CGI directory, which for the standard Apache distribution is:

```
/apache/cgi-bin
```

For the Red Hat Linux distribution, the CGI directory is:

```
/home/httpd/cgi-bin
```

That's all you need to do - proceed to the section describing how to [install and set up WebLink's Network Service Daemon \(NSD\)](#).

Note that WebLink's CGI modules are also available for NT/95/98 platforms (**mgwcgi.exe** and **nph-mgwcgi.exe**). Note also that because CGI is an open standard, WebLink's CGI modules will work with any Web server.

## Installing and Operating WebLink's Network Service Daemon

In order for the two Web connectivity options to operate, the WebLink NSD must be up and running. If necessary, create the following directory directly below the system root:

```
/cachesys/webblink
```

From the distribution, copy the NSD executable (**mgwmt** for UNIX and **mgwmt.exe** for NT/95/98) into this directory.

To start the NSD type:

```
./mgwmt
```

Before retiring to the background, the NSD will display various lines of text indicating the running configuration. It will show the TCP port number dedicated to this service, which is, by default, port number **7037**. This can be overridden by starting the service as follows:

```
./mgwmt <port_no>
```

where **port\_no** is the TCP port number of your choice.

All startup messages for this command can be suppressed using the **-s** qualifier. For example, if you wish to silently start the NSD from a script invoked when the system is booted use:

```
/cachesys/webblink/mgwmt -s [port_no]
```

To close down WebLink's NSD, issue the following command (UNIX systems):

```
kill -TERM `cat /cachesys/webblink/wlserver.pid`
```

This will close down WebLink in an orderly manner - WebLink will close down all the open connections to Caché and release all its system resources before terminating. Do not use the 'kill -9' command to terminate WebLink's NSD! For NT/95/98, follow the instructions given on the screen.

All errors will be reported in WebLink's event log (i.e. **mgw.log**). This file will be created and maintained in the **/cachesys/webblink** directory. WebLink's configuration file **mgw.ini** will also reside in this directory.

The NSD will also create the following file:

```
/cachesys/webblink/wlserver.ini
```

This file is extremely important because it tells the clients how to communicate with the NSD. In this context, the clients are the WebLink module contained within Apache and the WebLink CGI modules invoked by Apache. It is, therefore, essential that this file is not deleted or moved. It is also important that the Apache processes can read this file. Set the privileges accordingly, bearing in mind the UNIX user under which your Apache server is operating.

## Calling WebLink through Apache

Having started the WebLink NSD, you are ready to start Apache and run WebLink applications. The order in which these services are started does not actually matter.

If you chose the WebLink module approach, point your browser at:

`http://<ip_address>/scripts/mgwms32.dll`

- you should see the WebLink Systems Management forms. If you chose the CGI route, point your browser at:

`http://<ip_address>/cgi-bin/nph-mgwcgi`

- You can choose to run via both methods! You will have noticed that two CGI modules are supplied - **mgwcgi** and **nph-mgwcgi**. These are, in fact, the same. The **nph** prefix is the CGI convention for instructing the Web server not to parse the HTTP headers (i. e. non-parsed-headers). The Web server will simply forward your headers on to the browser.

WebLink and WebLink-based applications usually assume the responsibility for sending a full HTTP header. Under CGI, the first line of the header (the request status line) has traditionally been the responsibility of the server-side of the CGI interface. Some Web servers are strict about this - Apache is one such example. Apache expects the first header line sent by your application to be the **Content-type** line and not the request status line (e.g. HTTP/1.0 200 OK) - it will return an error if it parses the header and does not detect a **Content-type** line first. Using the 'nph' version of the CGI module alleviates this problem. Not all Web servers are so strict about this - Microsoft's Web server does not object to receiving a full HTTP header via mgwcgi.exe.



## Network Service Daemon

- [NSD Architecture](#)
- [NSD Features](#)
- [Deciding Which Version to Use](#)
- [Installing and Operating the NSD](#)

Caché WebLink provides Web connectivity to Caché and Open M databases by using the high-performance Web server APIs supplied by Microsoft (ISAPI) and Netscape (NSAPI). The close relationship between WebLink and its hosting Web server provides high performance, because WebLink runs in the same address space as the Web server.

However, in order to bring the benefits of WebLink to a wider range of Web servers and alternative Web architectures, the WebLink Network Service Daemon (NSD) is provided in this version of Caché WebLink.

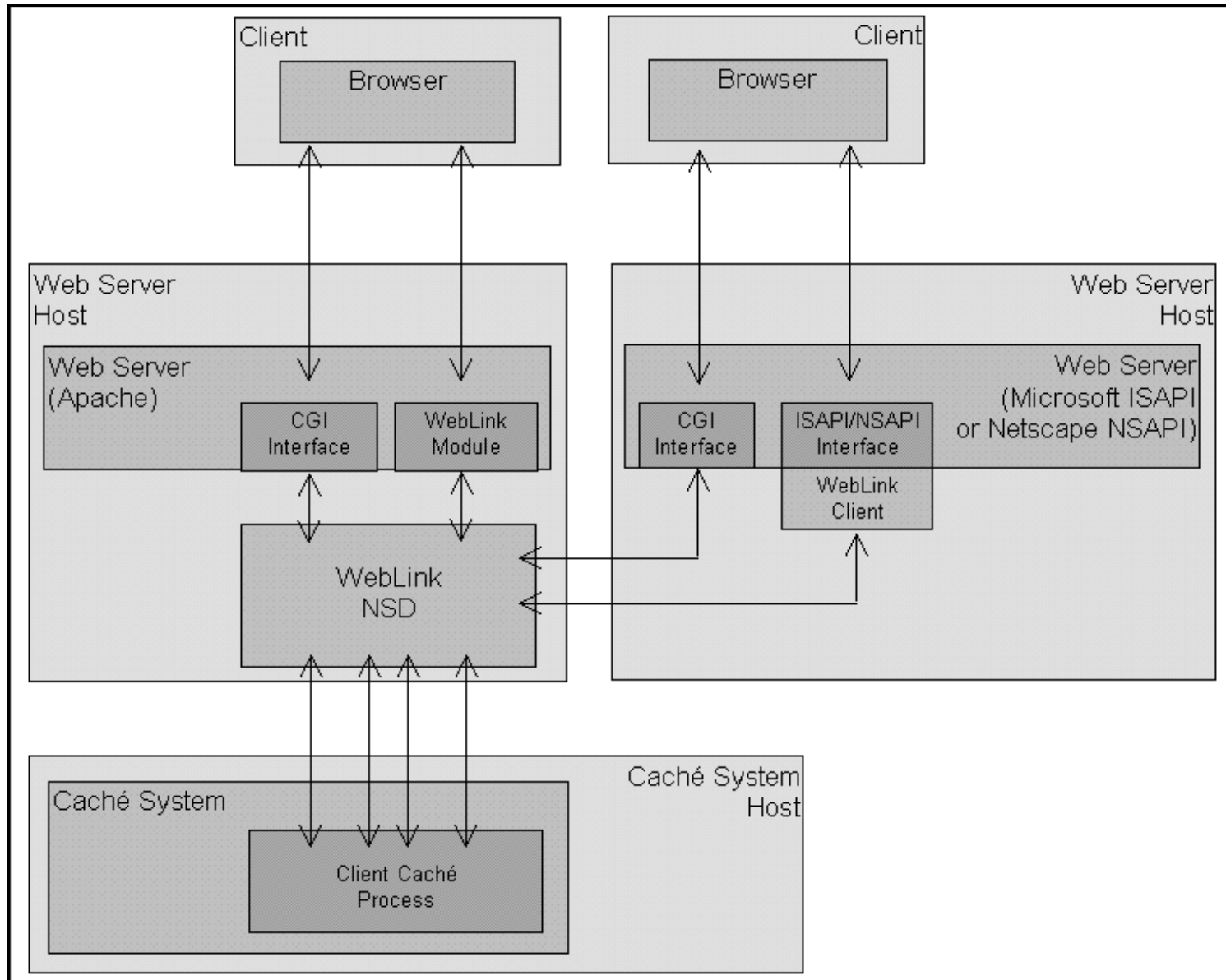
**Note:** The use of the NSD is optional. Therefore, in this chapter, the term *standard WebLink* is used for a site that is not using the NSD, while *WebLink NSD* is used for a site using the NSD.

### NSD Architecture

The WebLink NSD offers exactly the same functionality as the standard WebLink libraries, but delivers its services in a way that is totally independent of the Web server. This "stand-alone" version of WebLink allows you to retain the high-performance characteristics of the standard version while giving the extra flexibility described below, such as support for CGI.

The interface between the Web server and WebLink NSD has been kept as simple as possible. This simplicity makes WebLink eminently accessible to interpretive scripting languages (such as Perl) or straightforward shell scripting.

The following diagram illustrates the WebLink NSD architecture.





## NSD Features

The key features that NDS offers are:

- WebLink NSD is accessible via CGI scripts and executables. Because the WebLink NSD's CGI executable is small and simple, it starts up quickly. WebLink NSD maintains a persistent pool of Caché processes ready for use, so there is no additional overhead in connecting to the database. This design provides a high-performance CGI solution.
- Support Web servers that do not offer a multi-threaded API.
- Microsoft (ISAPI) and Netscape (NSAPI) Web servers can use the NSD service, in addition to the tailor-made standard WebLink solutions.
- Existing WebLink applications do not need to be changed, because they will operate as-is with both standard WebLink and WebLink NSD implementations.
- WebLink's configuration procedures remain the same, because the Systems Management pages are the same.
- As indicated in the diagram, WebLink NSD need not be installed on the same machine as the Web server. Although WebLink NSD is optimized to operate on the same machine as the Web server, there are at least two circumstances where the ability to operate on a separate machine is invaluable:
  - Supporting WebLink applications through independent Internet Service Providers (ISPs). Some ISPs are reluctant to accept WebLink as an add-on to their Web server and also provide local support for the Caché system to host the application. However, by using the WebLink NSD model, both WebLink NSD and Caché can be supported off-site™. Standard CGI would then be the mechanism for communicating between the ISP's Web server and WebLink NSD.
  - Supporting a Web server running in an operating environment not currently supported by WebLink (e.g., Netware). WebLink connectivity can be brought to such servers by installing WebLink NSD on a supported platform (such as UNIX or NT). The Web server running in the unsupported environment could then communicate with WebLink NSD via CGI.
- More than one Web server can be supported by the same WebLink NSD installation, as shown in the diagram.
- WebLink NSD can be managed independently of the Web server, and vice versa. For example, WebLink NSD can be taken out of service temporarily without disturbing the Web server. Likewise, the Web server can be taken out of service temporarily without disturbing WebLink NSD. This feature is particularly important if WebLink NSD is providing support for multiple Web servers.

## Deciding Which Version to Use

There is no conflict between standard WebLink and the WebLink NSD alternative. The standard (non-NSD) version is a more appropriate solution for small to medium sites using Microsoft or Netscape Web servers and not using CGI scripts. Keep in mind that you can start with the standard WebLink configuration and later add the NSD service.

Users of Web servers that do not support ISAPI or NSAPI must use WebLink NSD.

Users who prefer to access WebLink via CGI must also use WebLink NSD.

## Installing and Operating the NSD

The procedure is the same for both UNIX and NT/95/98.

If necessary, create the following directory directly below the system root:

```
/cachesys/weblink
```

From the distribution, copy the NSD executable (**mgwmt** for UNIX and **mgwmt.exe** for NT/95/98) into this directory.

To start the NSD type:

```
./mgwmt
```

Before retiring to the background, the NSD will display various lines of text indicating the running configuration. It will show the TCP port number dedicated to this service, which is, by default, port number **7037**. This can be overridden by starting the service as follows:

```
./mgwmt <port_no>
```

where *port\_no* is the TCP port number of your choice.

All startup messages for this command can be suppressed using the **'-s'** qualifier. For example, if you wish to 'silently' start the NSD from a script invoked when the system is booted use:

```
/cachesys/weblink/mgwmt -s [port_no]
```

To close down WebLink's NSD, issue the following command (UNIX systems):

```
kill -TERM `cat /cachesys/weblink/wlserver.pid`
```

This will close down WebLink in an orderly manner - WebLink will close down all the open connections to Caché and release all its system resources before terminating. Note that you should not use the 'kill -9' command to terminate WebLink's NSD.

For Windows NT/95/98, follow the instructions given on the screen.

All errors will be reported in WebLink's event log (i.e., '**mgw.log**'). This file will be created and maintained in the '**/cachesys/weblink**' directory. WebLink's configuration file '**mgw.ini**' will also reside in this directory.

The NSD will also create the following file:

`/cachesys/weblink/wlserver.ini`

This file is extremely important because it tells the 'clients' how to communicate with the NSD. In this context, the 'clients' are the WebLink module contained within Apache and the WebLink CGI modules invoked by Apache or any other CGI-compliant web server. It is, therefore, essential that this file is not deleted or moved. It is also important that the Weblink clients can read this file. Set the privileges accordingly, bearing in mind the UNIX user under which your participating WebLink clients are operating.



# Supplementary Installation and Configuration Information

- [Installing Caché WebLink Manually](#)
  - [Customizing Netscape Web Servers](#)
  - [Customizing Microsoft Web Servers](#)
- 

## Installing Caché WebLink Manually

1. Download the Caché WebLink self-extracting archive (for Windows 95/98 and NT) or the tar file (for UNIX) from the InterSystems [WebLink Web page](#). InterSystems recommends saving the UNIX archive in an empty temporary directory.
2. On a **Windows** system, double-click on the .exe file you downloaded. This begins the install process.  
On a **UNIX** system untar the file and run ./Setup.
3. Among the things you can specify in the setup program is which components to install and where to install them. If you do not have a Web server installed, the installation procedure places all the components for Caché WebLink in whatever directory you selected. The default location is **\Cachesys\WebLink** on the same drive as your windows directory or, if you have Caché installed, the same drive as Caché.

The Web server files are stored in the **dll** subdirectory according to processor chip.

- If you are using an intel processor the directory name is **i386**.
- If you are using an alpha processor, the directory name is **alpha**.

The routines are stored in the **routines** subdirectory of the user specified directory.

**Note:** The **OpenVMS** save routine MGW.RO can be downloaded directly to your OpenVMS system in ASCII or transferred from your UNIX or Windows system.

4. Copy or move the **mgw.ini** file (if supplied) into the active Microsoft Windows directory. If this file isn't supplied, WebLink will write a default copy when it is started for the first time.
5. Create a directory called scripts in the Web server directory if one does not already exist. Be sure that the permissions are set to executable by world.
6. For a Microsoft Web server, copy or move the dynamic link library **mgwms32.dll** to the scripts directory.  
For a Netscape Web server, copy or move the dynamic link library **mgwns20.dll** (Netscape version 2.x) or **mgwns30.dll** (Netscape version 3.x) into the scripts directory. **Note:** Netscape FastTrack version 3.x should use **mgwns20.dll**.

**Caution:** For Netscape Web servers, Caché WebLink requires modifications to your Netscape configuration files. The installation program displays the lines it needs to add and which configuration files contains those lines. Caché WebLink does this automatically. If these lines conflict with your current configuration, you should [manually customize your Netscape Web server](#).

## IBM AIX Netscape 2.0 patch

Install the patch to enable shared libraries, Shared Library Hookable Symbols/6000 PRPQ (slhs PRPQ). The patch is available from the [IBM FTP site \(ftp://service.boulder.ibm.com/aix/products/netscape/tools\)](ftp://service.boulder.ibm.com/aix/products/netscape/tools).

AIX 4.2 does support shared libraries, but the version is not compatible with Caché WebLink.

## Customizing Netscape Web Servers

Netscape Web servers require customization to use Caché WebLink. Note that this customization is unnecessary for Microsoft's Internet Information Server.

If you are using the Windows 95 or Windows NT server, installation will prompt you with the changes and you can review them and accepted them if they do not conflict with your existing configuration. If you have modified your configuration, you may want to make the changes by hand. If you are using the UNIX server, you must make the changes by hand.

**Note:** In all the code cited below, *servername* is the name you gave the Web server during the Web server installation.

Select the type of Netscape server you are going to modify.

- [Netscape Enterprise Server for NT](#)
- [Netscape FastTrack Server for NT](#)
- [Netscape Enterprise Server for UNIX](#)
- [Netscape FastTrack Server for UNIX](#)

## Netscape Enterprise Server for Windows 95/NT

1. Create a scripts directory within the Netscape server directory tree. The examples below show the default directory names. If you did not install in the default directory, substitute the directory you did install into where appropriate.

```
drive:\Netscape\Server\https-servername\scripts
```

2. Include the Caché WebLink object libraries.  
Find the **obj.conf** file, usually located in

```
drive:\Netscape\Server\https-servername\config\.
```

At the beginning of the **obj.conf**, just after the comment lines (comments are preceded by #) add:

```
Init fn=load-modules shlib=C:/Netscape/Server/https-servername/scripts/mgwns20.dll  
funcs=CacheWebLink
```

Substitute **mgwns20.dll** with **mgwns30.dll** as appropriate.  
At the end of the file add:

```
<Object ppath="*/scripts/*">Service Method=(Get|Head|Post) fn=CacheWebLink</Object>
```

3. Change the line in the configuration file called **mime.types** and generally located in

```
\Netscape\Server\https-servername\config\  
from:      type=text/html exts=htm,html  
to         type=text/html exts=htm,html,dll
```

## Netscape FastTrack Server for Windows 95/NT

1. Create a scripts directory within the Netscape server directory tree. The examples below show the default directory names. If you did not install in the default directory, substitute the directory you did install into where appropriate.

```
drive:\Netscape\Server\httpd-servername\scripts
```

2. Include the Caché WebLink object libraries  
Find the **obj.conf** file usually located in

```
drive:\Netscape\Server\httpd-servername\config\
```

At the beginning of the **obj.conf**, just after the comment lines (comments are preceded by #) add:

```
Init fn=load-modules shlib=C:/Netscape/Server/httpd-servername/scripts/mgwns20.dll  
funcs=CacheWebLink
```

Substitute **mgwns20.dll** with **mgwns30.dll** as appropriate.  
At the end of the file add:

```
<Object ppath="*/scripts/*">  
Service Method=(Get|Head|Post) fn-CacheWebLink  
</Object>
```

3. Change the line in the configuration file called **mime.types** and generally located in

```
from: \Netscape\Server\httpd-servername\config\  
type=text/html exts=htm,html  
to  
type=text/html exts=htm,html,dll
```

## Netscape Enterprise Server for UNIX

1. Create a scripts directory within the Netscape server directory tree. The examples below show the default directory names. If you did not install in the default directory, substitute the directory you did install into where appropriate.

```
/usr/ns-home/https-servername/scripts
```

2. Install the shared library mgwns.so into the scripts subdirectory created in step 1.
3. Find the **obj.conf**. It is usually located in

```
/usr/ns-home/https-servername/config .
```

Change the reference from:

```
mgwns20.dll (or mgwns30.dll)  
to:  
mgwns20.so (or mgwns30.so)
```

4. Change the line in the configuration file called **mime.types** and generally located in

```
from: /usr/ns-home/https-servername/config  
type=text/html exts=htm,html  
to  
type=text/html exts=htm,html,dll
```

## Netscape FastTrack Server for UNIX

1. Create a scripts directory within the Netscape server directory tree. The examples below show the default directory names. If you did not install in the default directory, substitute the directory you did install into where appropriate.

```
/usr/ns-home/httpd-servername/scripts
```

2. Install the shared library mgwns.so into the scripts subdirectory created in step 1.
3. Find the **obj.conf**. It is usually located in

```
/usr/ns-home/httpd-servername/config .
```

Change the reference from:

```
mgwns20.dll (or mgwns30.dll)  
to:  
mgwns20.so (or mgwns30.so)
```

4. Change the line in the configuration file called **mime.types** and generally located in

```
      /usr/ns-home/httpd-servername/config  
from:  type=text/html  exts=htm,html  
to     type=text/html  exts=htm,html,dll
```

## Customizing Microsoft Web servers

When using a Netscape browser with a Microsoft server, the browser may not correctly interpret the Caché WebLink DLLs. To fix this:

1. Open the registry editor (Start|Run type regedit OK).
2. Select HKEY\_LOCAL\_MACHINE|SYSTEM|CurrentControlSet|Services|W3SVC|Parameters|ScriptMap
3. Select Edit|New|String Value and type **.mgw** and press <ENTER>.
4. Double click the New Entry and enter the name of the Web server scripts directory followed by mgwms32.dll, the name of the WebLink dll. Generally the default will be: c:\inetpub\scripts\mgwms32.dll .
5. Restart the Web server.
6. You can now access WebLink using: **http://127.0.0.1/scripts/WebLink.mgw**



# InterSystems Corporation

## Copyright Notification

Copyright © InterSystems Corporation, 1999  
All rights reserved.

These documents contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

*M/SQL®*, *M/PACT®*, and *M/NET®* are registered trademarks, and *InterSystems™*, *Caché™*, *Caché WebLink™*, *Caché WebLink Developer™*, *Caché SQL™*, *ISM™*, *DTM™*, *DT-MAX™*, *DT Windows™*, *DSM™*, and *DASL™* are trademarks of InterSystems Corporation.

*Microsoft®* is a registered trademark and *Windows™* and *Windows NT™* are trademarks of Microsoft Corporation.

For Support questions about any InterSystems products, contact the InterSystems Worldwide Support Center:

**Phone:** US: +1 617 621-0700 Europe: +44 (0) 1753 830-077

**Fax:** US: +1 617 374-9391 Europe: +44 (0) 1753 861-311

**Internet Contact:** [support@intersys.com](mailto:support@intersys.com)

**[InterSystems FTP Site](#)**

**[InterSystems Web Site](#)**

**European BBS:** +44 (0) 1753-853-534