IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

ORACLE CORPORATION and ORACLE U.S.A. INC.,	)	
Plaintiffs/Counterclaim Defendants,	)	C.A. No. 06-414-SLR
v.	)	**JURY TRIAL DEMANDED**
EPICREALM LICENSING, LP,	)	**PUBLIC VERSION**
Defendant/Counterclaim Plaintiff.	)	

SECOND DECLARATION OF DR. DAVID FINKEL IN SUPPORT OF EPICREALM'S OPPOSITIONS TO ORACLE'S MOTIONS FOR SUMMARY JUDGMENT OF NONINFRINGEMENT AND INVALIDITY

VOLUME IV of V

EXHIBITS 40 to 46

Richard L. Horwitz (#2246)
David E. Moore (#3983)
POTTER ANDERSON & CORROON LLP
Hercules Plaza, 6th Floor
1313 N. Market Street
Wilmington, DE 19899
Tel: (302) 984-6000
rhorwitz@potteranderson.com
dmoore@potteranderson.com

Attorneys for Defendant/Counterclaim Plaintiff epicRealm Licensing, LP

OF COUNSEL:

Harry J. Roper
George S. Bosy
Aaron A. Barlow
Patrick L. Patras
David R. Bennett
Paul D. Margolis
Benjamin J. Bradford
Emily C. Johnson
JENNER & BLOCK
330 N. Wabash Avenue
Chicago, IL 60611-7603
Tel: (312) 923-8305

Dated: August 21, 2008
Public Version Dated: August 29, 2008
880482 / 31393 / Oracle

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

ORACLE CORPORATION and ORACLE U.S.A. INC.,	)	
Plaintiffs,	)	C.A. No. 06-cv-414 (SLR)
v.	)	
EPICREALM LICENSING, LP,	)	
Defendant.	)	

SECOND DECLARATION OF DR. DAVID FINKEL IN SUPPORT OF EPICREALM'S OPPOSITIONS TO ORACLE'S MOTIONS FOR SUMMARY JUDGMENT OF NONINFRINGEMENT AND INVALIDITY

I, David Finkel, declare as follows:

1. I am currently the Co-Director of the Computer and Communications Networks program at Worcester Polytechnic Institute. I have been a professor of Computer Science since 1991. I submit this declaration in support of EpicRealm's Oppositions to Oracle's Motions For Summary Judgment of Noninfringement and Invalidity.

2. I have been involved in studying, evaluating, and teaching about computer networks, network software, and the World Wide Web for over 15 years, and have authored numerous articles in the field. Additionally, I have taught undergraduate and graduate courses in the areas of computer networks and computer communications since 1989. Furthermore, I have taught several courses that specifically focus on Internet technology and the World Wide Web.

3. The Accused Oracle Application Server Products literally infringe Claim 1 of the '554 Patent under both parties' constructions. (*See* 2d Finkel Decl. Ex. 1 at 1-20.)

4. The Accused Oracle Application Server Products literally meet the "intercepting" limitation under either party's construction. (*See* 2d Finkel Decl. Exs. 1, 2.)

5. The "Web server" in all of the Accused Oracle Products processes some, but not all requests.

6. All software always goes through its "same iterative process," *i.e.*, it always follows its sequence of instructions. The software code does not change.

7. In Leaf, all requests are immediately forwarded on from the so-called "Web server" to transaction gateways; the "Web server" in Leaf could not process requests itself. Thus, a request could not be "intercepted" at the "Web server" in Leaf because there is no determination of whether the "Web server" or a "page server" should process the request—the

"Web server" in Leaf would never process a request itself. Therefore, Leaf did not disclose "intercepting."

8. The Accused Oracle Application Server Products literally meet the "releasing" limitation under either party's construction because the "Web server" (OHS) is released to process "other requests." (*See* 2d Finkel Decl. Exs. 1, 2.)

Redacted

Redacted

12. The Accused Oracle Application Server Products literally meet the "dispatcher" limitation under both its ordinary meaning and Oracle's construction. (*See* 2d Finkel Decl. Ex. 1, 2.)

13. Mod_oc4j meets Oracle's construction of "dispatcher" because it is a software program that determines which page server should process a dynamic Web page generation request.

14. Source code is written in modular fashion. For example, within one piece of software there can be one section of code that performs the "Web server" functionality and another section of code that performs the "dispatcher" functionality. Therefore, a request is routed from the "Web server" to the "dispatcher" when the request is routed from the code that performs the "Web server" functionality to the code that performs the "dispatcher" functionality.

15. The "Database Connector" is a Dynamically Linked Library (".dll") that runs in the same executable as the IIS 1.0 "Web server."

16. The Accused Oracle Database Products literally infringe claim 1 of the '554 Patent under both parties' constructions. (*See* 2d Finkel Decl. Ex. 3 at 1-21.)

Redacted

Redacted

22. Security attributes and privileges can come from a dynamic Web page request.

23. The Accused Oracle Web Cache Products literally infringe Claim 1 of the '554 Patent under both parties' constructions. (*See* 2d Finkel Decl. Ex. 5 at 1-19.)

24. The "Web server" (Web Cache) processes some but not all requests.

25. Web Cache is released to process other requests under either party's construction of "releasing." When Web Cache routes a request to a "page server," it is released to process

other requests under both parties' constructions in the same two ways—the same two ways that the "Web server" of the Accused Oracle Application Server Products are released.

Redacted

28. Merely having a "page server" process the request does not necessarily mean that the "releasing" limitation is met. For example, a system could be designed to have the "Web server" process a request in parallel with a "page server," *e.g.*, to help ensure a prompt response to a Web client even if the "page server" subsequently became slow or had a problem.

Redacted

31. The Accused Oracle Products also literally infringe the asserted claims that depend from Claim 1 of the '554 Patent. (*See* 2d Finkel Decl. Exs. 1, 3, 5.)

32. The Accused Oracle Products also literally infringe Claims 9 and 10 of the '554 Patent. (*See* 2d Finkel Decl. Exs. 1, 3, 5.)

33. The Accused Oracle Products also literally infringe Claim 11 of the '554 Patent. (*See* 2d Finkel Decl. Ex. 1, 3, 5.)

34. The Accused Oracle Products also literally infringe Claims 2 and 16 of the '335 Patent. (*See* 2d Finkel Decl. Ex. 2, 4, 6.)

35. Oracle provides various manuals to its customers instructing them how to use the Accused Oracle Products in an infringing manner.

36. The Accused Oracle Products infringe the asserted claims under the doctrine of equivalents. The Accused Oracle Products infringe because the differences between the Accused Oracle Products and the asserted claims are insubstantial from the perspective of a person of ordinary skill in the relevant art.

37. Nowhere in the specification does it discuss that "intercepting a request" requires that the Web server distinguish between dynamic and static content. Nor would one skilled in the art understand that the claims of the patents require such a distinction. Oracle Web Cache makes a determination of whether it will process a request itself (*e.g.*, satisfy the request with data stored in its cache) or it will send the request to an application server for processing.

38. The partitioned architecture of the claimed invention involves adding "page servers" which are capable of processing dynamic Web page requests. The "page servers" alleviate the processing burden placed on the "Web server" by processing some or all of the dynamic Web page requests. When a "page server" processes a request it "frees" or "releases" the "Web server" to process other requests because the "Web server" no longer has to expend resources processing that request. .

39. In the patented invention, the "dispatcher" can reside on a different machine than the "Web server" or the "dispatcher" can reside on the same machine as the "Web server." Regardless of where the "dispatcher" is located, it performs the same function. When the "dispatcher" resides on the same machine as the "Web server," some of the advantages of the invention will not be enjoyed. Namely, the Web server will not be able to offload the processing of the requests when a request is routed to the "dispatcher." The Web server, however, may still be able to offload the processing of a request when the request is routed from the "dispatcher"

residing on the Web server machine to a "page server." Thus, the purpose of the "dispatcher" is to "dispatch" requests, not to unburden the "Web server."

Redacted

41. The Oracle Web Agent does not make an informed selection of which page server should process the request. When the Oracle Web Agent determines which "page server" (Oracle7 Server) should process a request, it does not make a selection between Oracle7 Servers that are capable of processing the request. Rather, only one Oracle7 server could process the request. Oracle concedes that making an "informed ***selection***" requires, at least, choosing from more than one page server that is capable of processing the request. Thus, the Oracle Web Agent does not make an "informed selection."

42. The Oracle Web Agent also does not make a selection based on information indicating which page server can more efficiently process a request. Because only one Oracle7 server can process a given request, the information used by the Oracle Web Agent does not indicate the relative efficiencies of the Oracle7 servers. Instead, the information only indicates

which Oracle7 server to connect to and how to connect to that server. Thus, the Oracle Web Agent does not use information indicating which page server can more efficiently process a request in deciding which page sever should process a request.

43. The term "dynamic information" refers to information that can only be established during the execution of a program. Dynamic information does not refer to information that can be established prior to the execution of the program. Put another way, dynamic information is of a nature that can only be determined during runtime execution.

44. SQL*Net is remote data access software. Its fundamental purpose is to allow clients and servers to access Oracle databases from remote locations. SQL*Net is not concerned with the management of dynamic Web page requests. In fact, the term "Web page" is never used in the "Understanding SQL*Net 2.2" reference. Load balancing of connection requests has little to do with metric-based load balancing of dynamic Web page requests.

45. SQL*Net routes requests for database connections which are not the same as dynamic Web page requests as recited in all of the asserted claims, which are a type of "work requests." A request for a database connection attempts to establish a connection between a client and a server; the response to a request for a database connection will not be a dynamic Web page. By comparison, a dynamic Web page request typically results in a dynamic Web page in response and cannot establish a connection between a client and a database.

46. When a client is connecting to a multi-threaded database using SQL*Net 2.2 or SQL*Net 2.3, the request for a database connection sent from the client to the database is a wholly different request for a connection than the request originally received by SQL*Net.

47. One way of implementing the selection of the "least called" server is a round-robin algorithm.

48. Oracle 1.0 does not anticipate any of the asserted claims.

49. Claim 4 requires that the "page server" maintain a connection cache to said one or more data sources. Claim 5 requires that the processing step performed by the "page server" also includes the step of logging into one or more data sources. Oracle 1.0 does not anticipate Claims 4 and 5 of the '554 Patent for the additional reason that it cannot maintain a connection cache or log into one or more data sources. According to Oracle, the "page server" in Oracle 1.0 is the Oracle7 server, *i.e.* an Oracle database server. The data sources in Oracle 1.0 are the Oracle7 database. The Oracle7 server does not need to establish a connection to an Oracle7 database in order to retrieve data from the database because they are located on the same machine.

50. Oracle argues that Claim 5 is met because the "dispatcher" (the Oracle Web Agent) logs into the Oracle7 server. However, Claim 5 states that the "processing" step includes logging into one or more data sources and the Oracle Web Agent does not perform the "processing" step of Claim 1. Indeed, Oracle contends that the processing step is performed by the Oracle7 server, not the Oracle Web Agent.

51. Oracle 2.0 does not anticipate any of the asserted claims.

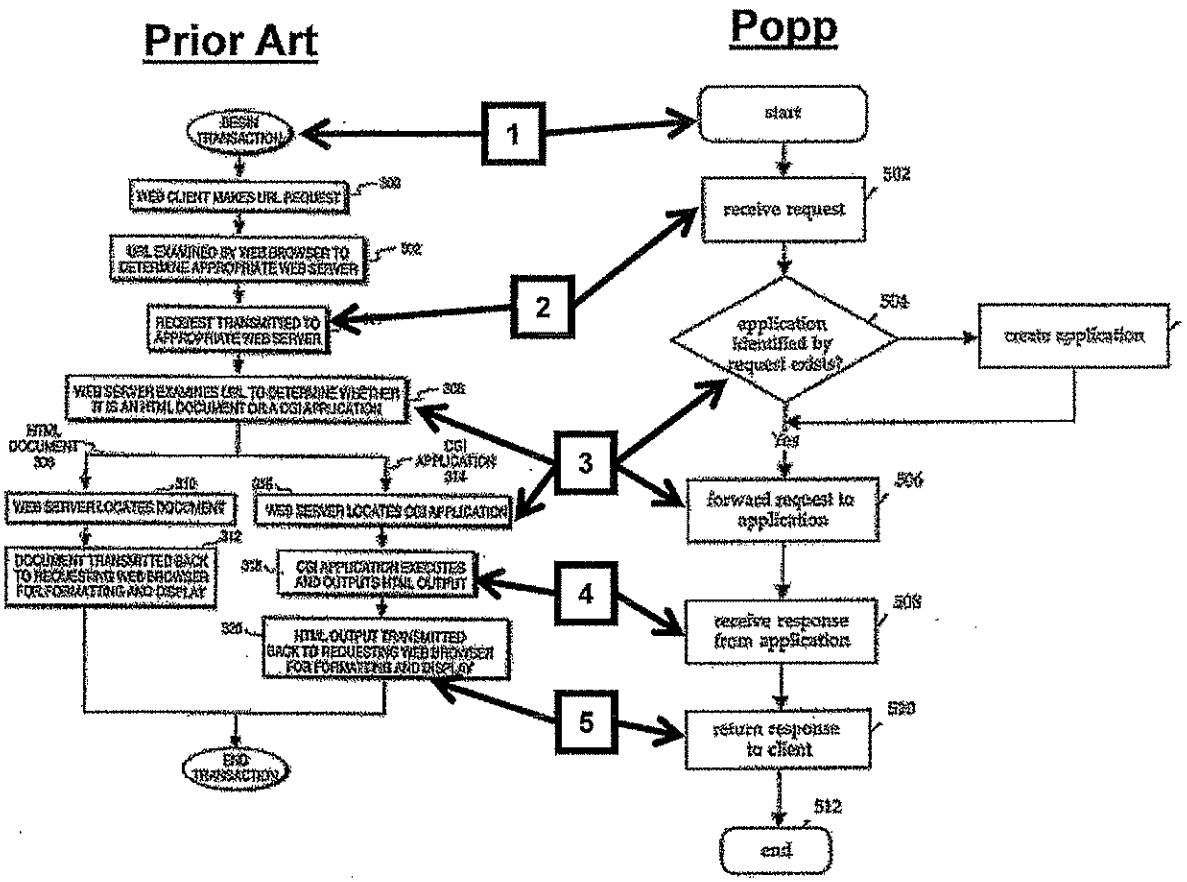
52. Oracle's proposed construction for page server is "page generating software, running on a processor separate from that of the Web server, that generates dynamic Web pages." However, Oracle's alleged "page server" in Oracle 2.0 is a WRBX that runs on the same machine (and therefore the same processor) as the Oracle 2.0 "Web server."

53. Oracle 2.0 does not anticipate Claim 4 for the additional reason that it does not disclose the "page server maintaining a connection cache to said one or more data sources." Under the ordinary meaning of "connection cache," as understood by one skilled in the art, maintaining a connection cache involved a cache of connections to a database that can be

checked out and reused by numerous processes. When a PL/SQL Agent stays connected to a database, there is neither a cache of connections nor can that connection be shared among various processes. Thus, it fails to meet the ordinary meaning of "connection cache."

54. Popp does not anticipate any of the asserted claims.

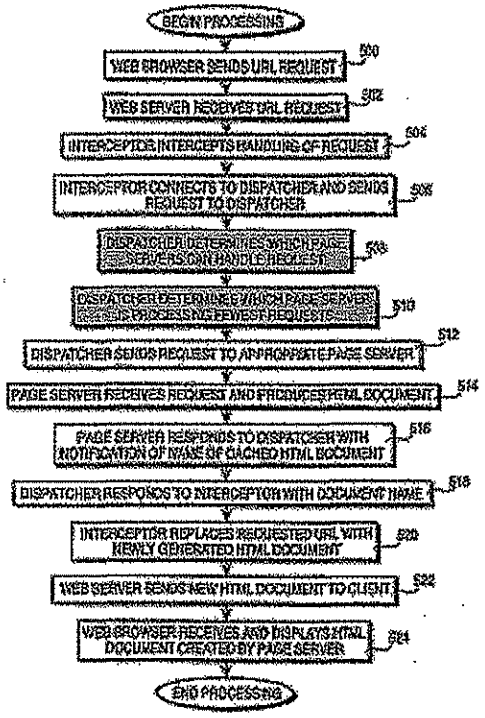
55. Below on the left is a flow chart describing how a request is processed in the prior art Web server disclosed by the '554 Patent and on the right is a flow chart from Popp describing how it processes a request:



nearly identical manner as the prior art disclosed by the patents in suit and considered by the

examiner. Specifically, Popp only determined if the application identified by a request existed, and if it did, forwarded the request to that application.

56. By comparison, in the flow diagram below depicting an embodiment of how the invention of the patents in suit processed a request, the "page server" is selected based on a two-step process (Steps 508 and 510 highlighted in gray below): (1) determine "which page servers can handle request;" and (2) determine "which page server is processing fewest requests" (*i.e.*, which "page server" can more efficiently process the request).



57. Popp's approach to Web site architecture is very similar to Oracle 1.0. In both cases, the primary concern was routing Web page requests to an application that could process the request. Neither Popp, nor Oracle 1.0, gave any consideration to determining which instance of an application "can *more efficiently* process the request." In fact, Oracle 1.0 and Popp are nearly identical to the prior art described in the '554 Patent specification.

58. By comparison, as shown in the embodiment illustrated in Figure 5 of the '554 Patent, the "page server" is selected using steps 508 ("dispatcher determines which page servers can handle request") and 510 ("dispatcher determines which page server is processing fewest requests"), *i.e.*, which "page server" can more efficiently process the request.

59. The CGI interface program in Popp (the alleged "dispatcher") does not make an "informed selection" of which page server should process a request. When the CGI interface program selects an application to process a request, it only determines which application can process a request because there is only one application that can process each request.

60. The CGI interface also does not make a selection based on information indicating which "page server" can more efficiently process a request. Because only one application can process a given request, the information used by the CGI interface does not indicate the relative efficiencies of the applications.

61. The information used by the CGI interface program in Popp is not "dynamic information maintained about page servers." The information relating to which application can process a request and on which server the application is executing is static information because it can be determined before runtime execution.

62. The CGI interface also does not make an "informed selection" because it does not use any information "to make a reasonable or accurate determination as to which [page server] could more efficiently process the request."

63. Dienst does not anticipate any of the asserted claims.

64. Dienst is not concerned with determining which "page server" can more efficiently process a request. Instead, it is concerned with locating the only "page server" that can process a request.

65. Dienst does not disclose "dispatching" for many of the same reasons that Oracle 1.0 and Popp do not disclose "dispatching." All these references are focused on locating the one and only "page server" that can process a given request.

66. The alleged "dispatcher" in Dienst does not make an "*informed selection*" of which page server should process a request. In Dienst, only one "page server" (Dienst Server) contains all the works for a specific publisher. Thus, when the alleged "dispatcher" selects a Dienst Server to process a request, it does not make a selection, but rather locates the only Dienst Server that contains that publisher's works.

67. The alleged "dispatcher" in Dienst also does not make a selection based on information indicating which "page server" can more efficiently process a request. The alleged "dispatcher" only uses information on which Dienst Server is capable of responding to a given request. Because there is only one Dienst Server that can process a request, there is no information on which Dienst Server can more efficiently process that request.

68. The information used by the CGI interface program is not dynamic information maintained about "page servers" indicating which "page server" can more efficiently process a request. Each Dienst Server downloads a listing of the locations of the other Dienst Servers so that it can search the other servers for works. This information is not dynamic information maintained about "page servers" indicating which "page server" can more efficiently process a request.

69. In my opinion, a person of ordinary skill in the art would have a Bachelor's degree in computer science or equivalent, with three to five years of technical experience in software implementation, or equivalent.

70. There is no "dispatching" under either party's proposed construction because the job of the alleged "dispatcher" is limited to locating the one "page server" that can process the given request.

71. Garland does not cure the "dispatching" deficiency in Oracle's invalidity arguments. The "Dispatcher Server" in Garland does not "examine" or "analyze" the request, as required by both parties' "dispatching" constructions. The "Dispatcher Server" also does not send the request to the selected server as required by both parties' "dispatching" constructions.

72. None of the asserted claims are obvious over the combinations of references argued by Oracle.

73. Oracle has not proved by clear and convincing evidence that Garland in combination with any of Oracle's four other references would have rendered obvious to one of ordinary skill in the art any of the twelve asserted claims at the time of the respective inventions.

74. Because Dienst is concerned with providing access to a distributed digital library over the Web, it does not disclose many of the limitations added by the dependent claims of the '554 Patent, which are focused on features to further improve the performance of a Web site. Oracle argues that it would be obvious to combine Dienst with the teachings of Oracle 1.0 or Oracle 2.0 to meet these claims. In addition, none of these references discloses "dispatching." Their combination would not change that fact. Not only does Oracle fail to set forth its basis for claiming that it would be obvious to combine these references, but the secondary considerations argue against such combinations.

75. Further, it would not be obvious to combine Dienst, a reference on providing access to a distributed digital library with Oracle 1.0 or Oracle 2.0. In fact, it would be impossible to use the "page servers" of Oracle 1.0 or Oracle 2.0 to implement Dienst.

76. The information used by JDBC to identify connections that are authenticated with the proper user credentials, user name and password and to identify connections that have the necessary striping can come from the request.

77. I am over the age of eighteen, and, if called to do so, I am prepared and competent to testify on the matters presented in this declaration.

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct. Executed on August 21, 2008, in Texarkana, Texas.

Dr. David Finkel

AUG-21-2008 23:04 P.17

TOTAL P.17

TABLE OF CONTENTS

Exh.	*Document*
1	Claim Chart U.S. Patent No. 5,894,554 – The Accused Oracle Application Server Products
2	Claim Chart U.S. Patent No. 6,415,335 – The Accused Oracle Application Server Products
3	Claim Chart U.S. Patent No. 5,894,554 – The Accused Oracle Database Products
4	Claim Chart U.S. Patent No. 6,415,335 – The Accused Oracle Database Products
5	Claim Chart U.S. Patent 5,894,554 – Accused Oracle Web Cache Products
6	Claim Chart U.S. Patent No. 6,415,335 – The Accused Oracle Web Cache Products
7	Oracle WebServer User's Guide Release 1.0 Part Number A349862" (153)
8	Popp et al. U.S. Patent No. 6,249,291 (195)
9	Oracle WebServer User's Guide Release 2.0.2 Production, Part No. A23646-2 (604/605)
10	Oracle WebServer User's Guide Release 2.0 Production, Part No. A23646-1 and Oracle WebServer 2.0 Technical Note
11	Functional Specification for Runtime Load Balancing in OCI Session Pools, 11gR1, 12/2/05, CHID00081-00105
12	Excerpt from deposition transcript of Lakshminarayanan Chidambaran, Feb. 29, 2008
13	Excerpt from deposition transcript of Carol Colrain, April 15, 2008
14	Excerpt from deposition transcript of Hal Hildebrand, Dec. 21, 2007
15	Excerpt from deposition transcript of Matthew Mayerson, April 15, 2008
16	Excerpt from deposition transcript of Mark Nelson, Feb. 22, 2008
17	Excerpt from deposition transcript of Douglas Surber, Feb. 28, 2008
18	Design Specification for OCI Runtime Connection Load Balancing in 11g, 12/14/05, ORCL01310468-01310498
19	Functional Specification for OFCGI, OHS, iAS, 11AS, 6/2/06, ORCL01209371-01209387

20 Functional Specification for Runtime Load Balancing in OCI Session Pools, 11gR1, 12/2/05, ORCL01166752-01166776

21 Functional Specification for Runtime Load Balancing, RDBMS, 10gR2, 4/1/04, ORCL01060148-01060193

22 Functional/Design Specification for Runtime Work Request Load Balancing using RAC Service Metrics, JDBC, 10gR2, 5/1/04, ORCL01054844-01054858

23 Excerpt from Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide 10g (9.0.4), B10320-01 Sept. 2003, ORCL00337734-00338101

24 Excerpt from Oracle Application Server Containers for J2EE, Support for JavaServer Pages Developer's Guide 10g Release 2 (10.1.2) B14014-02, July 2005, ORCL00444343-00444558

25 Excerpt from Oracle Application Server Web Cache Administrator's Guide 10g Release 2 (10.1.2) B14045-01, Dec. 2004, ORCL00473184-00473739

26 Excerpt from Oracle Call Interface Programmer's Guide, 11g Release 1 (11.1), B28395-02, Oct. 2007

27 Excerpt from Oracle Call Interface Programmer's Guide, 11g Release 1 (11.1), B28395-03, May 2008

28 Excerpt from Oracle Database Administrator's Guide, 10g Release 2 (10.2), B14231-02, May 2006, ORCL01969895-01970763

29 Excerpt from Oracle HTTP Server Administrator's Guide 10g (9.0.4), B10381-02, March 2004, ORCL00372061-00372362

30 Oracle Real Application Clusters 11g: An Oracle Technical White Paper, April 2007, ORCL01972320-01972333

31 Excerpt from Oracle Web Cache Administration and Deployment Guide, Release 1.0.2, A86722-01, Nov. 21, 2000, ORCL00107955-00108126

32 Excerpt from Oracle XML DB Developer's Guide 11g Release 1 (11.1), October 2007, ORCL01972631-01973585

33 Excerpt from Oracle9i Application Serer: Availability, Scalability and Manageability of J2EE and Web Cache Clusters, An Oracle White Paper, May 2002 ORCL00013356-00013398

34 Excerpt from Oracle9i Application Server Performance Guide, Release 2 (9.0.2), April 2002, A95102-02, ORCL00847081-00847294

35 Excerpt from Oracle9i Application Server, An Oracle White Paper, Sept. 2002, ORCL00010730-00010792

36 Excerpt from Oracle9i Application Server, Best Practices, Release 1 (v1.0.2.2) Sept. 2001 A95201-01, ORCL00190234-00190397

37 Oracle9i Application Server, ORCL00110683

38 Excerpt from Oracle9i JDBC Developer's Guide and Reference Release 2 (9.2) March 2002, ORCL00306997-00307580

39 Excerpt from Oracle9iAS Web Cache Administration and Deployment Guide, Release 2 (9.0.2), May 2002, Part No. A95404-02, ORCL00272342-00272887

40 Excerpt from Oracle9iAX Web Cache Administration and Deployment Guide, Release 2.0.0, Oct. 10, 2001, A90372-04, ORCL00189864-00190165

41 Presentation, Mod_oc4j Load Balancing in iAS 9.0.4, ORCL00018115-00018150

42 Excerpt from Plaintiff's Responses to Defendants' First Set of Interrogatories (1-6)

43 Tanenbaum, A.S., *Computer Networks*, Fourth Edition. Prentice-Hall PTR, Upper Saddle River, NJ, 2003.

44 Oracle WebServer User's Guide Release 1.0, A349862 ORCL00941713-00941960 (MAC003037-003284) (*Ref 153*)

45 US 6,651,108, Popp et al., MAC003907-003927 (*Ref. 194*)

46 US 6,249,291, Popp et al., MAC003928-003968 (*Ref. 195*)

47 Oracle WebServer User's Guide Release 2.0.2 Production, Part No. A23646-2, 1996, ORCL00000782-00001035 (*Ref. 604*)

48 Oracle WebServer 2.0 Technical Note, March 1996, ORCL00000761-00000781 (*Ref. 605*)

49 Oracle and the Internet, Using Oracle in the Emerging Electronic Infrastructure of Business and Society, July 1995, ORCL00937779-00937806

50 Excerpt from Understanding SQL*Net, Release 2.3, A42484-1, 1996, ORCL01808597-01808820

51 The Rush to the Net, May 1996, ORCL01606514-01606526

52 Web Request Broker API, Ver. 0.8, Feb. 12, 1996, ORCL01606852-01606867

53 Excerpt from deposition transcript of Katrina Montinola, Feb. 19, 2008

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

CERTIFICATE OF SERVICE

I, David E. Moore, hereby certify that on August 29, 2008, the attached document was electronically filed with the Clerk of the Court using CM/ECF which will send notification to the registered attorney(s) of record that the document has been filed and is available for viewing and downloading.

I further certify that on August 29, 2008, the attached document was Electronically Mailed to the following person(s):

Mary B. Graham
Morris, Nichols, Arsht & Tunnell LLP
1201 North Market Street
P.O. Box 1347
Wilmington, DE 19899
MGraham@MNAT.com

James G. Gilliland
Igor Shoiket
Townsend and Townsend and Crew LLP
Two Embarcadero Center
Eighth Floor
San Francisco, CA 94111-3834
OracleEpicrealm@townsend.com

Theodore T. Herhold
Robert J. Artuz
Eric M. Hutchins
Eric A. Mercer
Joseph A. Greco
Nitin Gupta
Townsend and Townsend and Crew LLP
379 Lytton Avenue
Palo Alto, CA 94301
OracleEpicrealm@townsend.com

Chad E. King
Townsend and Townsend and Crew LLP
1200 Seventeenth Street
Suite 2700
Denver, CO 80202
OracleEpicrealm@townsend.com

/s/ David E. Moore
Richard L. Horwitz
David E. Moore
Potter Anderson & Corroon LLP
Hercules Plaza – Sixth Floor
1313 North Market Street
P.O. Box 951
Wilmington, DE 19899-0951
(302) 984-6000
rhorwitz@potteranderson.com
dmoore@potteranderson.com

Exhibit 40

Oracle9*i*AS Web Cache

Administration and Deployment Guide

Release 2.0.0

October 10, 2001

Part No. A90372-04

ORACLE

Oracle9*i*AS Web Cache Administration and Deployment Guide, Release 2.0.0

Part No. A90372-04

Copyright © 2000, 2001, Oracle Corporation. All rights reserved.

Primary Author: Deborah Steiner

Contributors: Jose Alvarez, Steve Andrew, Jesse Anton, Christine Chan, Ajay Dsai, Jay Feenan, Patrick Fry, Pramodini Gattu, Helen Grembowicz, Hideaki Hayashi, Martin Littlecott, Larry Jacobs, Lars Klevan, Xiang Liu, Rajiv Mishra, Jordan Parker, Marcin Porwit, Charles Qi, Parthiban Thilagar, William Wright, Jean Zeng, Tie Zhong, and Yuhui Zhu

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and Oracle9*i* and Oracle8*i* are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

1

Introduction to Oracle Web Cache

This chapter describes the performance barriers faced by Web sites and introduces the technology which can provide a complete caching solution.

This chapter contains these topics:

- What is the Big Picture for Caching?
- Oracle's Solution to Web Site Performance Issues
- How Web Caching Works
- Benefits of Web Caching
- Features of Oracle Web Cache

What is the Big Picture for Caching?

The e-business model creates new performance requirements for Web sites. To carry out electronic business successfully, Web sites must protect against poor response time and system outages caused by peak loads. Slow performance translates into lost revenue.

Many high-volume Web sites try to counter this problem by adding more application Web servers to their existing architecture. As more users access these Web sites, more and more application Web servers will have to be added. In short, the manageability costs associated with adding application Web servers often outweigh the benefits.

Static caches and content distribution services can provide some relief. However, these solutions are unable to serve content that is dynamically generated.

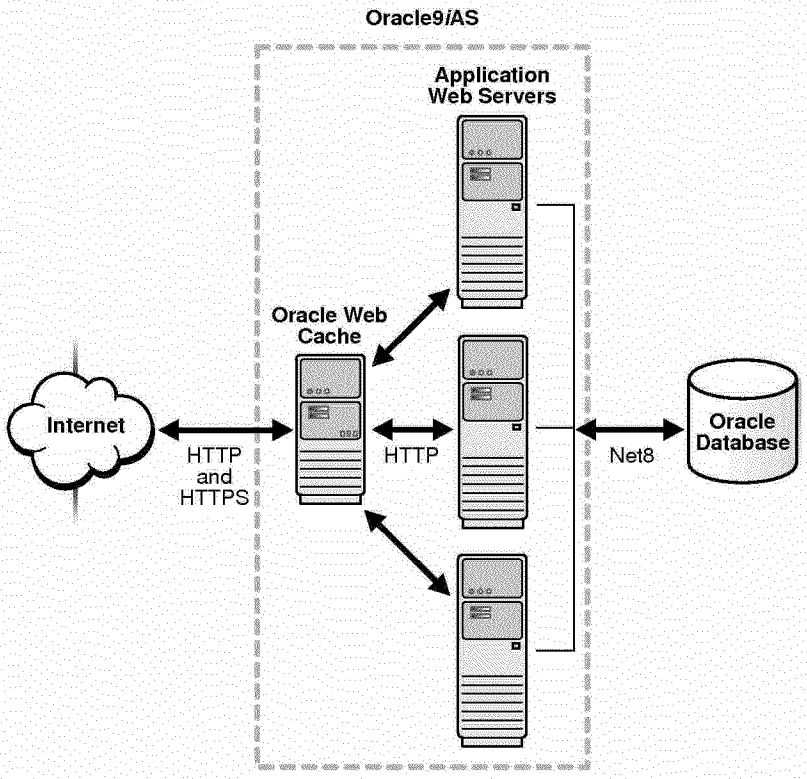
Oracle's Solution to Web Site Performance Issues

Faced with these performance challenges, e-businesses need to invest in more cost-effective technologies and services to improve the performance of their sites. Oracle offers Oracle Web Cache to help e-businesses manage Web site performance issues. Oracle Web Cache is a content-aware server accelerator, or **reverse proxy server**, that improves the performance, scalability, and availability of Web sites that run on Oracle9*i* Application Server and Oracle8*i*.

By storing frequently accessed URLs in memory, Oracle Web Cache eliminates the need to repeatedly process requests for those URLs on the application Web server. Unlike legacy proxy servers that handle only static documents, Oracle Web Cache caches both static and dynamically generated content from one or more application Web servers. Because Oracle Web Cache is able to cache more content than legacy proxies, it provides optimal performance by greatly reducing the load on application Web servers.

Figure 1–1 shows the basic architecture. Oracle Web Cache sits in front of application Web servers, caching their content, and providing that content to Web browsers that request it. When Web browsers access the Web site, they send HTTP protocol or HTTPS protocol requests to Oracle Web Cache. Oracle Web Cache, in turn, acts as a virtual server to the application Web servers. If the requested content has changed, Oracle Web Cache retrieves the new content from the application Web servers. The application Web servers may retrieve their content from an Oracle database.

Figure 1–1 Oracle Web Cache Architecture



How Web Caching Works

To Web browsers, Oracle Web Cache acts as the virtual server for application Web servers. You configure Oracle Web Cache with the same IP address that is registered for a site's domain name and the application Web servers' host names. This configuration enables Web browsers to communicate with Oracle Web Cache rather than application Web servers when accessing a Web site.

Figure 1–2 on page 1-5 shows how Web caching works. Oracle Web Cache has an IP address of `144.25.190.240` and the application Web server has an IP address of `144.25.190.245`. The steps for browser interaction with Oracle Web Cache follow:

1. A browser sends a request to a Web site named `www.company.com`.

   This request in turn generates a request to Domain Name System (DNS) for the IP address of the Web site.

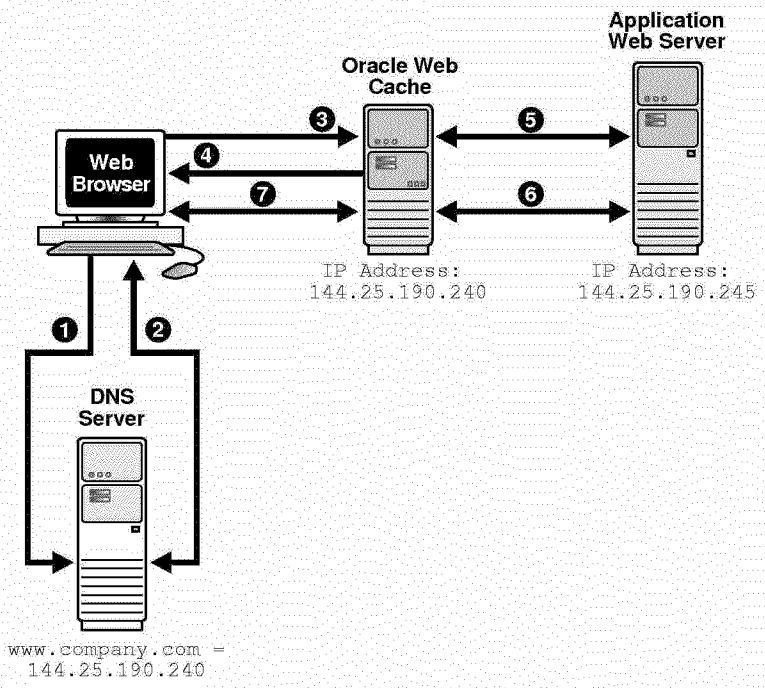
2. DNS returns the IP address of Oracle Web Cache, that is, `144.25.190.240`.
3. The browser sends the request for the Web page to Oracle Web Cache, `144.25.190.240`.
4. If the requested content is in its cache, then Oracle Web Cache sends the content directly to the browser. This is called a **cache hit**.

   Note: Dynamic content is generated by the application Web server and then returned to Oracle Web Cache before being passed to the browser.

5. If Oracle Web Cache does not have the requested content or the content is stale or invalid, it hands the request off to the application Web server. This is called a **cache miss**.
6. The application Web server sends the content through Oracle Web Cache.
7. Oracle Web Cache sends the content to the client and makes a copy of the page in cache.

   Note: A page stored in the cache is removed when it becomes invalid or outdated, as described in "Cache Freshness and Performance Assurance" on page 2-4.

Figure 1–2 Web Server Acceleration



Benefits of Web Caching

Web caching provides the following benefits for Web sites:

- Performance

  Running on inexpensive hardware, Oracle Web Cache can increase the throughput of a Web site by several orders of magnitude. In addition, Oracle Web Cache significantly reduces response time to browser requests by storing documents in memory and by serving compressed versions of documents to browsers that support the GZIP encoding method.

- Scalability

  In addition to unparalleled throughput, Oracle Web Cache can sustain thousands of concurrent browser connections, meaning that visitors to a site see fewer application Web server errors, even during periods of peak load.

- High Availability

  Oracle Web Cache supports content-aware (Layer 4 - Layer 7) load balancing and failover detection. These features ensure that cache misses are directed to the most available, highest-performing Web server in the cluster. Moreover, a patent-pending capacity heuristic guarantees performance and provides surge protection when application Web server load increases.

- Cost Savings

  Better performance, scalability and availability translates into cost savings for Web site operators. Because fewer application Web servers are required to meet the challenges posed by traffic spikes and denial of service attacks, Oracle Web Cache offers a simple and inexpensive means of reducing a Web site's cost for each request.

- Network Traffic Reduction

  Most requests are resolved by Oracle Web Cache, reducing traffic to the application Web servers. The cache also reduces traffic to backend databases located on computers other than the application Web server.

Features of Oracle Web Cache

The main features of Oracle Web Cache make it a perfect caching service for e-business Web sites that host online catalogs, news services, and portals. These features include:

- Static and Dynamically Generated Content Caching
- Cache Invalidation
- Performance Assurance
- Surge Protection of Application Web Servers
- Load Balancing of Application Web Servers
- Security Features
- Administration
- Compression

Static and Dynamically Generated Content Caching

Oracle Web Cache uses cacheability rules to store documents. These rules fall into two categories:

- Rules for static content, such as GIF, JPEG, or static HTML files
- Rules for dynamically generated content created using technologies like Java Server Pages (JSP), Active Server Pages (ASP), PL/SQL Server Pages (PSP), Java Servlets, and Common Gateway Interface (CGI). Support of these technologies enables Oracle Web Cache to recognize rules for the following:
  - Multiple-version documents for the same URL, that is, the same URL with slightly different content
  - Session-aware rules for pages containing session information
  - Personalization rules for pages containing personalized greetings, such as "Welcome <Name>," and session-encoded URLs
- Pages that require content assembly of dynamic fragments

See Also:

- "Caching Dynamically Generated Content" on page 2-7 for further information about dynamically-generated content
- "Content Assembly and Partial Page Caching" on page 2-17

Cache Invalidation

Oracle Web Cache supports **invalidation** as a mechanism to keep its cache consistent with the content on the application Web servers, origin databases, or other dynamically generated means.

Administrators can invalidate cache content in one of two ways:

- Send an invalidation message to the computer running Oracle Web Cache

  When documents are invalidated and a browser requests them, Oracle Web Cache refreshes them with new content from the application Web server.

- Assign an expiration time limit to the documents

  When a document expires, Oracle Web Cache treats it like an invalid document, that is, if requested by a browser, it refreshes it with a updated content from the application Web server.

See Also: "Cache Freshness and Performance Assurance" on page 2-4 for further information about invalidation

Performance Assurance

When a large number of documents have been invalidated, the retrieval of a new documents can result in overburdened application Web servers.

To handle performance issues while maintaining cache consistency, Oracle Web Cache uses built-in **performance assurance heuristics** that enable it to assign a queue order to documents. These heuristics determine which documents can be served stale and which documents must be refreshed immediately. Documents with a higher priority are refreshed first. Documents with a lower priority are refreshed at a later time.

The queue order of documents is based on the popularity of documents and the validity of documents assigned during invalidation. If the current load and capacity of the application Web server is not exceeded, the most popular and least valid documents are refreshed first.

See Also: "Cache Freshness and Performance Assurance" on page 2-4 for further information about performance assurance

Surge Protection of Application Web Servers

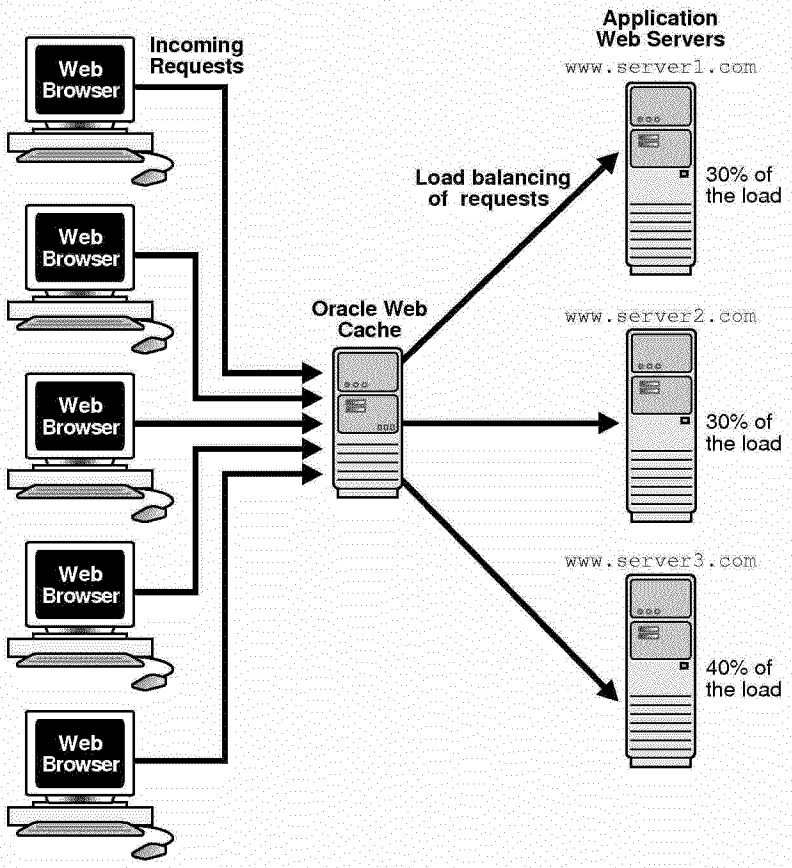
Oracle Web Cache passes requests for non-cacheable or stale documents to the application Web servers. To prevent an overload of requests on the application Web servers, Oracle Web Cache has a surge protection feature that enables you to set a limit on the number of concurrent requests that the application Web servers can handle. When the limit is reached, subsequent requests are queued to wait up to a maximum amount of time. If the maximum wait time is exceeded, Oracle Web Cache rejects the request and serves a site busy apology page to the Web browser that initiated the request.

Load Balancing of Application Web Servers

Most Web sites are served by multiple application Web servers running on multiple computers that share the load of HTTP and HTTPS requests. This feature enables Web sites to be built with a collection of servers for better scalability and reliability. Oracle Web Cache is designed to manage requests for up to 100 application Web servers. All requests that Oracle Web Cache cannot serve are passed to the application Web servers. Oracle Web Cache has a **load balancing** feature that distributes these requests over a set of application Web servers. To configure load balancing, you set the capacity (concurrent connections) for each application Web server. Based on the capacity assigned, Oracle Web Cache prescribes the relative percentage load of each application Web server.

Figure 1–3 shows three application Web servers, whereby 30 percent of traffic goes to `www.server1.com`, 30 percent goes to `www.server2.com`, and 40 percent goes to `www.server3.com`.

Figure 1–3 Load Balancing



See Also:

- "Task 3: Specify Web Site Settings" on page 5-5 for configuration details
- "Configuring Load Balancing and Failover" on page 7-3 for configuration details

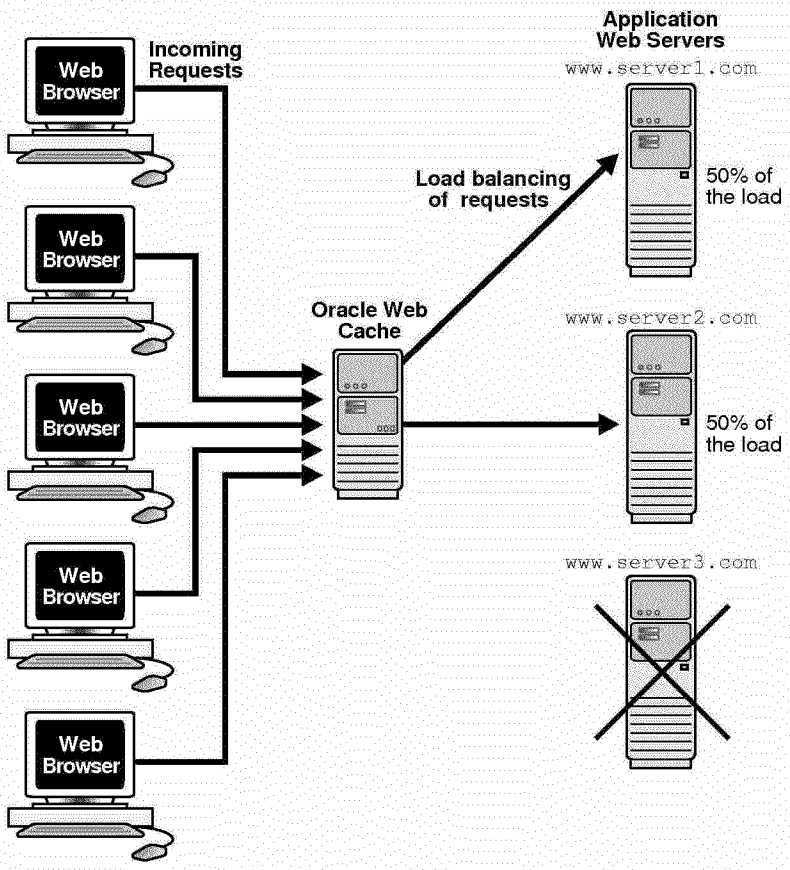
Backend Failover

After a specified number of continuous request failures, Oracle Web Cache considers an application Web server as failed. When an application Web server fails, Oracle Web Cache automatically distributes the load over the remaining application Web servers based on the remaining proportion. Oracle Web Cache polls the failed application Web server for its current up/down status until it is back online.

The **failover** feature is shown in Figure 1–4 on page 1-13. An outage of `www.server3.com`, which was receiving 40 percents of requests, results in 50 percent of the requests going to `www.server1.com` and 50 percent of requests going to `www.server2.com`. This is based on a 30 percent load for both `www.server1.com` and `www.server2.com`. If `www.server1.com` were to fail instead, based on a 30 percent load for `www.server2.com` and a 40 percent load for `www.server3.com`, 42.86 percent of the requests would go to `www.server2.com` and 57.14 percent of requests would go to `www.server3.com`.

When the failed server returns to operation, Oracle Web Cache will include it in the load mix as previously prescribed.

Figure 1–4 Failover



Application Web Server Binding

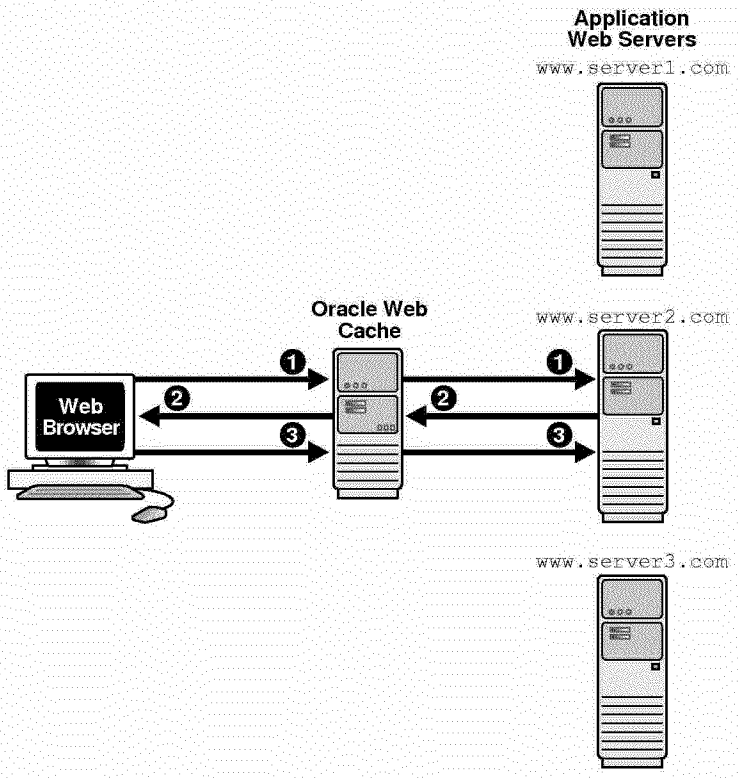
Oracle Web Cache supports Web sites that use session IDs and/or **session cookies** to bind user sessions to a given application Web server in order to maintain state for a period of time. To utilize the **session binding** feature, the application Web server itself must maintain state, that is, it must be stateful. Web sites bind user sessions by including session data in the HTTP header or body it sends to Web browsers in such a way that the browser is forced to include it with its next request. This data is transferred either with parameters embedded in the URL or cookies, which are text strings stored on the client.

Figure 1–5 on page 1-15 shows how Oracle Web Cache supports documents that use application Web server binding:

1. When a request first comes in, Oracle Web Cache uses load balancing to decide which application Web server to send it to. In this example, `www.server2.com` was chosen.
2. If the requested document requires application Web server binding, the application Web server sends the session information back to the browser through Oracle Web Cache in the form of a cookie or an embedded URL parameter.
3. Oracle Web Cache sends subsequent requests for the session to the application Web server that established the session, bypassing load balancing. In this example, `www.server2.com` handles the subsequent requests.

> **See Also:** "Binding a Session to an Application Web Server" on page 7-3 for configuration details

Figure 1–5 Application Web Server Binding



Security Features

Oracle Web Cache provides the following security-related features:

- Restricted Administration
- Secure Sockets Layer (SSL) Support

Restricted Administration

Oracle Web Cache restricts administration with the following features:

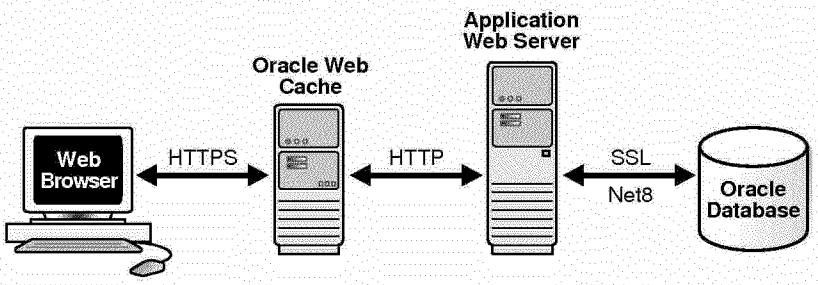
- Password authentication for administration and invalidation operations
- Control over which ports administration and invalidation operations can be requested from
- IP and subnet administration restrictions

Secure Sockets Layer (SSL) Support

The **Secure Sockets Layer (SSL)** protocol, developed by Netscape Corporation, is an industry-accepted standard for network transport layer security. SSL provides authentication, encryption, and data integrity, in a public-key infrastructure (PKI). By supporting SSL, Oracle Web Cache is able to cache pages for HTTPS requests.

As shown in Figure 1–6, you can configure Oracle Web Cache to receive HTTPS browser requests. In addition, you can secure connections between the application Web server and the database with SSL.

Figure 1–6 SSL for Secure Connections



ORCL00189911

Limitations: HTTPS support has the following limitations in this release:

- Oracle Web Cache does not provide authentication or access control.
- Oracle Web Cache does not support client-side certification.
- The HTTPS protocol is not supported between Oracle Web Cache and application Web server(s).

SSL interacts with the following entities:

- Certificate Authority
- Certificate
- Wallet

Certificate Authority A certificate authority (CA) is a trusted third party that certifies the identity of third parties and other entities, such as users, databases, administrators, clients, and servers. The certificate authority verifies the party identity and grants a certificate, signing it with the its private key. The Oracle Web Cache certificate must be signed by a CA.

Different CAs may have different identification requirements when issuing certificates. One may require the presentation of a user's driver's license, while others may require notarization of the certificate request form, or fingerprints of the requesting party.

The CA publishes its own certificate, which includes its public key. Each network entity has a list of certificates of the CAs it trusts. Before communicating with another entity, a given entity uses this list to verify that the signature on the other entity's certificate is from a known, trusted CA.

Network entities can obtain their certificates from the same or different CAs. By default, Oracle Advanced Security automatically installs trusted certificates from VeriSign, RSA, Entrust, and GTE CyberTrust when you install a new wallet (See: Wallet).

Certificate A certificate is created when a party's public key is signed by a trusted CA. A certificate ensures that a party's identification information is correct, and that the public key actually belongs to that party.

A certificate contains the party's name, public key, and an expiration date—as well as a serial number and certificate chain information. It can also contain information about the privileges associated with the certificate.

When a network entity receives a certificate, it verifies that it is a trusted certificate—one issued and signed by a trusted certificate authority. A certificate remains valid until it expires or is terminated.

Wallet A wallet is a transparent database used to manage authentication data such as keys, certificates, and trusted certificates needed by SSL. A wallet has an X.509 version 3 certificate, private key, and list of trusted certificates.

Security administrators use the Oracle Wallet Manager to manage security credentials on the Oracle Web Cache server. Wallet owners use it to manage security credentials on clients. Specifically, Oracle Wallet Manager is used to do the following:

- Generate a public-private key pair and create a certificate request for submission to a certificate authority.
- Install a certificate for the identity.
- Configure trusted certificates for the identity.

Note: Installation of Oracle9*i*AS Release 1.0.2.2 also installs Oracle Wallet Manager release 2.0.

See Also: *Oracle Wallet Manager User's Guide*

How SSL Works The authentication process between the browser and Oracle Web Cache consists of the following basic steps:

1. The browser initiates a connection to Oracle Web Cache by using HTTPS.
2. SSL performs the handshake between the browser and Oracle Web Cache.

At the commencement of an HTTPS network connection between a browser and Oracle Web Cache, an SSL handshake is performed. An SSL handshake between the browser and Oracle Web Cache includes the following actions:

- The browser and Oracle Web Cache establish which cipher suites to use.
- Oracle Web Cache sends its certificate to the browser, and the browser verifies that the Oracle Web Cache's certificate was signed by a trusted CA.
- The browser and Oracle Web Cache exchange key information using public key cryptography; based on this information, each generates a session key. All subsequent communications between the browser and the Oracle Web Cache is encrypted and decrypted by using this set of session keys and the negotiated cipher suite.

Administration

Oracle Web Cache provides a graphical user interface tool called **Oracle Web Cache Manager** that combines configuration and monitoring options to provide an integrated environment for configuring and managing Oracle Web Cache and the Web sites it caches for. With Oracle Web Cache Manager, you can easily:

- Configure Oracle Web Cache to cache for application Web servers
- Start and stop Oracle Web Cache
- Establish cacheability rules
- Monitor Oracle Web Cache and Web site performance
- Establish listening ports and security passwords

Compression

You can select to have Oracle Web Cache compress both cacheable and non-cacheable documents upon insertion into the cache for browsers. Because compressed documents are smaller in size, they are delivered faster to browsers with fewer round-trips, reducing overall **latency**. On average, Oracle Web Cache is able to compress text files by a factor of 4. For example, 300 KB files are compressed down to 75 KB.

Exhibit 41

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 42

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 43

Computer Networks

Fourth Edition

Andrew S. Tanenbaum

Vrije Universiteit
Amsterdam, The Netherlands



Prentice Hall PTR
Upper Saddle River, NJ 07458
www.phptr.com

Library of Congress Cataloging-in-Publication Data

Tanenbaum, Andrew S.
Computer networks / Andrew S. Tanenbaum.--4th ed.
p. cm.
Includes bibliographical references.
ISBN 0-13-066102-3
1. Computer networks. I. Title.
TK5105.5 .T36 2002
004.6--dc21 2002029263

Editorial/production supervision: *Patti Guerrieri*
Cover design director: *Jerry Votta*
Cover designer: *Anthony Gemmellaro*
Cover design: *Andrew S. Tanenbaum*
Art director: *Gail Cocker-Bogusz*
Interior Design: *Andrew S. Tanenbaum*
Interior graphics: *Hadel Studio*
Typesetting: *Andrew S. Tanenbaum*
Manufacturing buyer: *Maura Zaldivar*
Executive editor: *Mary Franz*
Editorial assistant: *Noreen Regina*
Marketing manager: *Dan DePasquale*



© 2003 Pearson Education, Inc.
Publishing as Prentice Hall PTR
Upper Saddle River, New Jersey 07458

Prentice Hall books are widely used by corporations and government agencies for training, marketing, and resale.

For information regarding corporate and government bulk discounts please contact:
Corporate and Government Sales (800) 382-3419 or corpsales@pearsontechgroup.com

All products or services mentioned in this book are the trademarks or service marks of their respective companies or organizations.

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

ISBN 0130661023

Text printed in the United States on recycled paper at Courier Westford in Westford, Massachusetts.

11th Printing October 2007

Pearson Education LTD.
Pearson Education Australia PTY, Limited
Pearson Education Singapore, Pte. Ltd.
Pearson Education North Asia Ltd.
Pearson Education Canada, Ltd.
Pearson Educación de Mexico, S.A. de C.V.
Pearson Education — Japan
Pearson Education Malaysia, Pte. Ltd.

6.1.3 Berkeley Sockets

Let us now briefly inspect another set of transport primitives, the socket primitives used in Berkeley UNIX for TCP. These primitives are widely used for Internet programming. They are listed in Fig. 6-5. Roughly speaking, they follow the model of our first example but offer more features and flexibility. We will not look at the corresponding TPDUs here. That discussion will have to wait until we study TCP later in this chapter.

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Figure 6-5. The socket primitives for TCP.

The first four primitives in the list are executed in that order by servers. The SOCKET primitive creates a new end point and allocates table space for it within the transport entity. The parameters of the call specify the addressing format to be used, the type of service desired (e.g., reliable byte stream), and the protocol. A successful SOCKET call returns an ordinary file descriptor for use in succeeding calls, the same way an OPEN call does.

Newly-created sockets do not have network addresses. These are assigned using the BIND primitive. Once a server has bound an address to a socket, remote clients can connect to it. The reason for not having the SOCKET call create an address directly is that some processes care about their address (e.g., they have been using the same address for years and everyone knows this address), whereas others do not care.

Next comes the LISTEN call, which allocates space to queue incoming calls for the case that several clients try to connect at the same time. In contrast to LISTEN in our first example, in the socket model LISTEN is not a blocking call.

To block waiting for an incoming connection, the server executes an ACCEPT primitive. When a TPDU asking for a connection arrives, the transport entity creates a new socket with the same properties as the original one and returns a file descriptor for it. The server can then fork off a process or thread to handle the connection on the new socket and go back to waiting for the next connection on the original socket. ACCEPT returns a normal file descriptor, which can be used for reading and writing in the standard way, the same as for files.

Now let us look at the client side. Here, too, a socket must first be created using the SOCKET primitive, but BIND is not required since the address used does not matter to the server. The CONNECT primitive blocks the caller and actively starts the connection process. When it completes (i.e., when the appropriate TPDU is received from the server), the client process is unblocked and the connection is established. Both sides can now use SEND and RECV to transmit and receive data over the full-duplex connection. The standard UNIX READ and WRITE system calls can also be used if none of the special options of SEND and RECV are required.

Connection release with sockets is symmetric. When both sides have executed a CLOSE primitive, the connection is released.

6.1.4 An Example of Socket Programming: An Internet File Server

As an example of how the socket calls are used, consider the client and server code of Fig. 6-6. Here we have a very primitive Internet file server along with an example client that uses it. The code has many limitations (discussed below), but in principle the server code can be compiled and run on any UNIX system connected to the Internet. The client code can then be compiled and run on any other UNIX machine on the Internet, anywhere in the world. The client code can be executed with appropriate parameters to fetch any file to which the server has access on its machine. The file is written to standard output, which, of course, can be redirected to a file or pipe.

Let us look at the server code first. It starts out by including some standard headers, the last three of which contain the main Internet-related definitions and data structures. Next comes a definition of *SERVER_PORT* as 12345. This number was chosen arbitrarily. Any number between 1024 and 65535 will work just as well as long as it is not in use by some other process. Of course, the client and server have to use the same port. If this server ever becomes a worldwide hit (unlikely, given how primitive it is), it will be assigned a permanent port below 1024 and appear on *www.iana.org*.

The next two lines in the server define two constants needed. The first one determines the chunk size used for the file transfer. The second one determines how many pending connections can be held before additional ones are discarded upon arrival.

After the declarations of local variables, the server code begins. It starts out by initializing a data structure that will hold the server's IP address. This data structure will soon be bound to the server's socket. The call to *memset* sets the data structure to all 0s. The three assignments following it fill in three of its fields. The last of these contains the server's port. The functions *htonl* and *htons* have to do with converting values to a standard format so the code runs correctly on both big-endian machines (e.g., the SPARC) and little-endian machines (e.g., the Pentium). Their exact semantics are not relevant here.

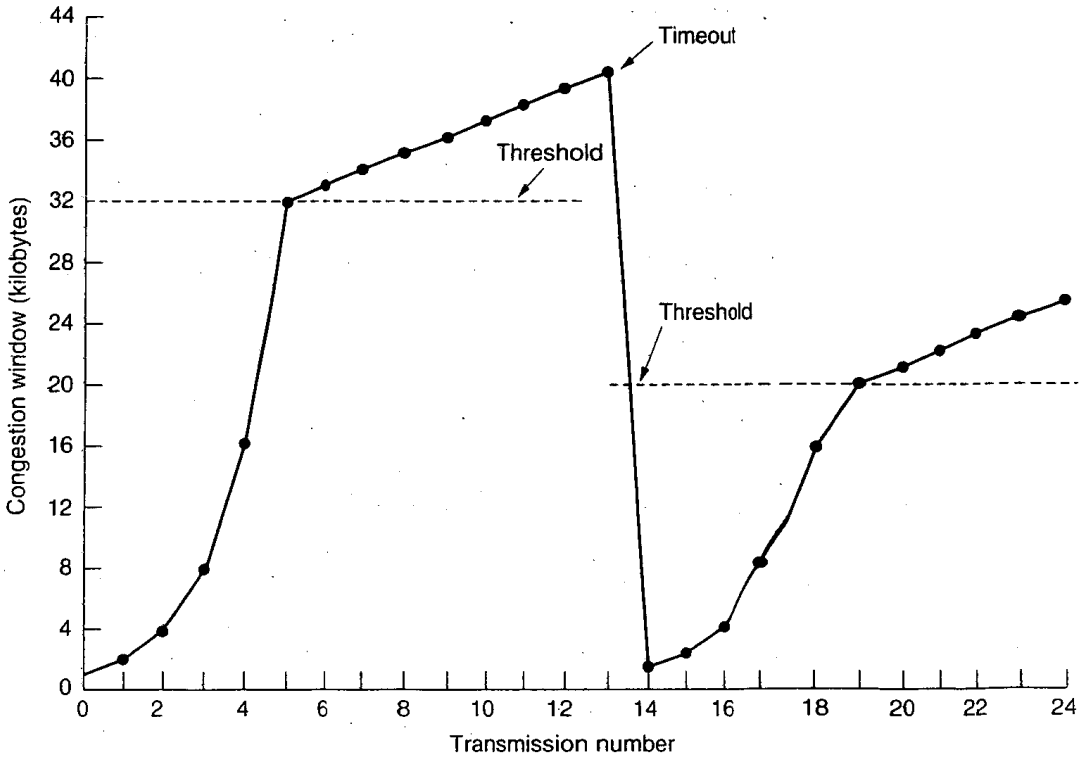


Figure 6-37. An example of the Internet congestion algorithm.

6.5.10 TCP Timer Management

TCP uses multiple timers (at least conceptually) to do its work. The most important of these is the **retransmission timer**. When a segment is sent, a retransmission timer is started. If the segment is acknowledged before the timer expires, the timer is stopped. If, on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted (and the timer started again). The question that arises is: How long should the timeout interval be?

This problem is much more difficult in the Internet transport layer than in the generic data link protocols of Chap. 3. In the latter case, the expected delay is highly predictable (i.e., has a low variance), so the timer can be set to go off just slightly after the acknowledgement is expected, as shown in Fig. 6-38(a). Since acknowledgements are rarely delayed in the data link layer (due to lack of congestion), the absence of an acknowledgement at the expected time generally means either the frame or the acknowledgement has been lost.

TCP is faced with a radically different environment. The probability density function for the time it takes for a TCP acknowledgement to come back looks more like Fig. 6-38(b) than Fig. 6-38(a). Determining the round-trip time to the destination is tricky. Even when it is known, deciding on the timeout interval is

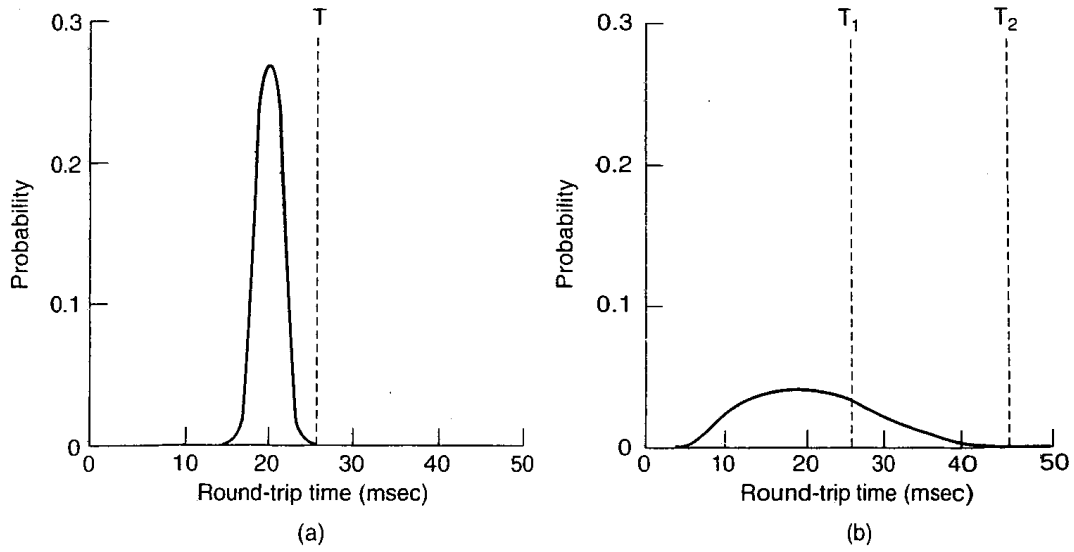


Figure 6-38. (a) Probability density of acknowledgement arrival times in the data link layer. (b) Probability density of acknowledgement arrival times for TCP.

also difficult. If the timeout is set too short, say, T_1 in Fig. 6-38(b), unnecessary retransmissions will occur, clogging the Internet with useless packets. If it is set too long, (e.g., T_2), performance will suffer due to the long retransmission delay whenever a packet is lost. Furthermore, the mean and variance of the acknowledgement arrival distribution can change rapidly within a few seconds as congestion builds up or is resolved.

The solution is to use a highly dynamic algorithm that constantly adjusts the timeout interval, based on continuous measurements of network performance. The algorithm generally used by TCP is due to Jacobson (1988) and works as follows. For each connection, TCP maintains a variable, *RTT*, that is the best current estimate of the round-trip time to the destination in question. When a segment is sent, a timer is started, both to see how long the acknowledgement takes and to trigger a retransmission if it takes too long. If the acknowledgement gets back before the timer expires, TCP measures how long the acknowledgement took, say, *M*. It then updates *RTT* according to the formula

$$RTT = \alpha RTT + (1 - \alpha)M$$

where α is a smoothing factor that determines how much weight is given to the old value. Typically $\alpha = 7/8$.

Even given a good value of *RTT*, choosing a suitable retransmission timeout is a nontrivial matter. Normally, TCP uses βRTT, but the trick is choosing β. In the initial implementations, β was always 2, but experience showed that a constant value was inflexible because it failed to respond when the variance went up.

In 1988, Jacobson proposed making β roughly proportional to the standard deviation of the acknowledgement arrival time probability density function so that a large variance means a large β, and vice versa. In particular, he suggested using the *mean deviation* as a cheap estimator of the *standard deviation*. His algorithm requires keeping track of another smoothed variable, D, the deviation. Whenever an acknowledgement comes in, the difference between the expected and observed values, $|RTT - M|$, is computed. A smoothed value of this is maintained in D by the formula

$$D = \alpha D + (1 - \alpha)\,|RTT - M|$$

where α may or may not be the same value used to smooth RTT. While D is not exactly the same as the standard deviation, it is good enough and Jacobson showed how it could be computed using only integer adds, subtracts, and shifts—a big plus. Most TCP implementations now use this algorithm and set the timeout interval to

$$\text{Timeout} = RTT + 4 \times D$$

The choice of the factor 4 is somewhat arbitrary, but it has two advantages. First, multiplication by 4 can be done with a single shift. Second, it minimizes unnecessary timeouts and retransmissions because less than 1 percent of all packets come in more than four standard deviations late. (Actually, Jacobson initially said to use 2, but later work has shown that 4 gives better performance.)

One problem that occurs with the dynamic estimation of RTT is what to do when a segment times out and is sent again. When the acknowledgement comes in, it is unclear whether the acknowledgement refers to the first transmission or a later one. Guessing wrong can seriously contaminate the estimate of RTT. Phil Karn discovered this problem the hard way. He is an amateur radio enthusiast interested in transmitting TCP/IP packets by ham radio, a notoriously unreliable medium (on a good day, half the packets get through). He made a simple proposal: do not update RTT on any segments that have been retransmitted. Instead, the timeout is doubled on each failure until the segments get through the first time. This fix is called **Karn's algorithm**. Most TCP implementations use it.

The retransmission timer is not the only timer TCP uses. A second timer is the **persistence timer**. It is designed to prevent the following deadlock. The receiver sends an acknowledgement with a window size of 0, telling the sender to wait. Later, the receiver updates the window, but the packet with the update is lost. Now both the sender and the receiver are waiting for each other to do something. When the persistence timer goes off, the sender transmits a probe to the receiver. The response to the probe gives the window size. If it is still zero, the persistence timer is set again and the cycle repeats. If it is nonzero, data can now be sent.

A third timer that some implementations use is the **keepalive timer**. When a connection has been idle for a long time, the keepalive timer may go off to cause

one side to check whether the other side is still there. If it fails to respond, the connection is terminated. This feature is controversial because it adds overhead and may terminate an otherwise healthy connection due to a transient network partition.

The last timer used on each TCP connection is the one used in the *TIMED WAIT* state while closing. It runs for twice the maximum packet lifetime to make sure that when a connection is closed, all packets created by it have died off.

6.5.11 Wireless TCP and UDP

In theory, transport protocols should be independent of the technology of the underlying network layer. In particular, TCP should not care whether IP is running over fiber or over radio. In practice, it does matter because most TCP implementations have been carefully optimized based on assumptions that are true for wired networks but that fail for wireless networks. Ignoring the properties of wireless transmission can lead to a TCP implementation that is logically correct but has horrendous performance.

The principal problem is the congestion control algorithm. Nearly all TCP implementations nowadays assume that timeouts are caused by congestion, not by lost packets. Consequently, when a timer goes off, TCP slows down and sends less vigorously (e.g., Jacobson's slow start algorithm). The idea behind this approach is to reduce the network load and thus alleviate the congestion.

Unfortunately, wireless transmission links are highly unreliable. They lose packets all the time. The proper approach to dealing with lost packets is to send them again, and as quickly as possible. Slowing down just makes matters worse. If, say, 20 percent of all packets are lost, then when the sender transmits 100 packets/sec, the throughput is 80 packets/sec. If the sender slows down to 50 packets/sec, the throughput drops to 40 packets/sec.

In effect, when a packet is lost on a wired network, the sender should slow down. When one is lost on a wireless network, the sender should try harder. When the sender does not know what the network is, it is difficult to make the correct decision.

Frequently, the path from sender to receiver is heterogeneous. The first 1000 km might be over a wired network, but the last 1 km might be wireless. Now making the correct decision on a timeout is even harder, since it matters where the problem occurred. A solution proposed by Bakne and Badrinath (1995), **indirect TCP**, is to split the TCP connection into two separate connections, as shown in Fig. 6-39. The first connection goes from the sender to the base station. The second one goes from the base station to the receiver. The base station simply copies packets between the connections in both directions.

The advantage of this scheme is that both connections are now homogeneous. Timeouts on the first connection can slow the sender down, whereas timeouts on the second one can speed it up. Other parameters can also be tuned separately for

Exhibit 44

Oracle® WebServer User's Guide

Release 1.0
Part No. A34986–2

ORACLE®

MAC003037

Oracle WebServer User's Guide, Release 1.0

Part No. A34986–2

Copyright © 1995 Oracle Corporation

All rights reserved. Printed in the U.S.A.

Contributing Authors: Mala Anand, Matthew Bookman, Martin Gruber, Magnus Lonnroth,Katrina Montinola, Mary Owen, and Joe Pistritto

Contributors: Arthur Gleckler, Walter Hudson, Kelly Ireland, Aileen Jaitin, Max Pollak, Sandy Venning, and Marcel Weekes

This software was not developed for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It is the customer's responsibility to take all appropriate measures to ensure the safe use of such applications if the programs are used for such purposes.

This software/ documentation contains proprietary information of Oracle Corporation; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

If this software/ documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of DFARS 252.227–7013, Rights in Technical Data and Computer Software (October 1988).

Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

If this software/ documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with " Restricted Rights", as defined in FAR 52.227–14, Rights in Data – General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free.

Oracle, SQL*Forms, SQL*DBA, SQL*Loader, SQL*Net and SQL*Plus are registered trademarks of Oracle Corporation.

PL/ SQL and Oracle7 are trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

Preface

Purpose

The Oracle WebServer User's Guide describes the Oracle WebServer and explains how to use and administer its various components.

Audience

This manual is designed for two distinct audiences: the programmer writing applications that publish Oracle data over the World Wide Web, and the administrator managing a commercial Web site that has deployed the Oracle WebServer.

How This Book is Organized

The following is a list of chapters, and the contents of each chapter:

Chapter 1 Overview

This chapter introduces the components of the Oracle WebServer, and explains how they work together.

Chapter 2 Introduction to HTML

This chapter provides basic information about HTML. The chapter also provides step by step instructions for creating an HTML file to be viewed on a Web browser.

Chapter 3 The Oracle Web Listener

This chapter explains how to use and administer the Oracle Web Listener. The chapter shows how the Oracle Web Listener handles requests for static and dynamic documents, and how it returns documents to the client.

Chapter 4 Introduction to the Oracle7 Server and to PL/SQL

This chapter provides a brief overview of how to use the Oracle7 Server, and how to use PL/ SQL, a language for developing applications for an Oracle database.

Chapter 5 Oracle Web Agent

This chapter describes the Oracle Web Agent, the main differentiating factor between the Oracle WebServer and other Web servers with database backends. The chapter also describes how to set up and manage the Oracle Web Agent.

Chapter 6 Oracle WebServer Developer's Toolkit

This chapter explains how to use special utilities that generate HTML tags within PL/ SQL procedures, thereby easing the development of dynamic HTML pages with PL/ SQL.

Chapter 7 The Oracle WebServer Administration Utility

This chapter explains how to use the Administration Utility, which is a set of Web pages that make it easy for you to create and configure Oracle Web Listeners and Oracle Web Agent services.

Appendix A Oracle WebServer Messages

This appendix describes the messages generated by the components ofthe Oracle WebServer. Foreach message that is caused by an error, it offers a cause and a suggested action.

> Note: If you are reading this manual on–line on the Web, you will notice a number of hypertext links to help you navigate *through the document. However, be aware that HTML does not* support a return to the origin of a link within a document. If you use a link to move to a different section of the document, you do not return to the anchor, but to the previous target that you reached through a link.

Conventions Used in This Manual

The notational conventions used in this document are described in the following table:

MAC003040

ORCL00941716

Feature	Explanation
monospace text	Text to be enterd by the user. Type text exactly as shown. Text typed on a PC is not case–sensitive, unless noted otherwise; text typed on a UNIX system is case sensitive. Monospace is also used when referring to text in syntax statements or example code.
italics	Italics represent a variable. Substitute an appropriate value. Italics are also used for emphasis, to draw attention to the first use of a technical term, and for names of documents.
[ ]	Brackets enclose optional items or indicate a function key. Do not type the brackets.
|	A vertical bar represents an "or" option among several options. You must enter one, and only one, of the options separated by the vertical bar. Do not enter the vertical bars themselves.
punctuation	Punctuation other than brackets and vertical bars must be entered in commands exactly as shown.
UPPERCASE	UPPERCASE characters within the text represent command names and HTML tags and attributes.

How to Use this Manual

Every reader of this manual should read Chapter 1, "Overview". This chapter gives a brief description of the Oracle WebServer and how it works.

If you are unfamiliar with HyperText Markup Language, see Chapter 2, "Introduction to HTML".

Administrators should read Chapter 3, "Oracle Web Listener," Chapter 5, "Oracle Web Agent," and Chapter 7 "The Oracle WebServer Administration Utility" to understand how these components work together, and how to administer them.

Programmers will be particularly interested in Chapter 5 "Oracle Web Agent," as well as Chapter 6, "Oracle WebServer Developer's Toolkit," which explains how to generate dynamic HTML documents in PL/ SQL. It lists all the functions and procedures provided by the Oracle

WebServer Developer's Toolkit to simplify the task of generating dynamic documents.

If you are unfamiliar with the Oracle7 Server and PL/ SQL, read the introduction to these two subjects in Chapter 4, "Introduction to the Oracle7 Server and to PL/ SQL".

Related Publications

The following books contain more information on the Oracle7 Server and PL/ SQL programming language:

- Oracle7 Server Concepts Manual

  This manual provides general information about the Oracle7 Server and how it works.

- Oracle7 Server Administrator's Guide

  This manual tells you how to administer the Oracle7 Server.

- Oracle7 Server Application Developer's Guide

  This manual tells you how to develop applications for the Oracle7 Server.

- Oracle7 Server Utilities User's Guide

  This manual tells you how to use SQL*DBA, Export, Import, and SQL*Loader, the utilities bundled with the Oracle7 Server.

- PL/SQL User's Guide and Reference Manual

  This manual gives you information about PL/ SQL, the Oracle7 procedural language extension to SQL.

- Oracle7 Server SQL Reference

  This manual gives you reference information about Oracle SQL commands and functions.

Your Comments Are Welcome

We value and appreciate your comments as an Oracle user and reader of the manuals. As we write, revise, and evaluate our documentation, your opinions are the most important input we receive. At the back of our printed manuals is a Reader's Comment Form, which we encourage you to use to tell us what you like and dislike about this manual or other Oracle manuals. If the form is not available, please use the following address or FAX number.

Oracle Internet Products Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA 94065
U.S.A.
FAX: 415–506–7200

MAC003043

ORCL00941719

[redacted]

Contents

Chapter 1

Overview 1–1
What is the Oracle WebServer? 1–2
Components of the Oracle WebServer 1–2
The Oracle Web Listener 1–3
The Oracle Web Agent 1–4
The Oracle WebServer Developer's Toolkit 1–4
Oracle7 Server 1–5
Summary 1–5

Chapter 2

Introduction To HTML 2–1
What is HTML? 2–2
How Are HTML Documents Created? 2–2
Getting Started 2–3
Document Structure 2–4
Body Tags 2–4
Summary of Basic HTML Tags 2–7
List Tags 2–8
Ordered Lists 2–8
Unordered Lists 2–9
Nested Lists 2–9
Definition Lists 2–10
Hypertext Linking 2–11
What is a URL? 2–11
Graphics within HTML Documents 2–14
Linking to Sections on Pages 2–15
Reviewing Changes to Your HTML Document 2–16

Adding Style to Your HTML Document . . . . . 2 – 16
Special HTML Tags . . . . . 2 – 17
Tables . . . . . 2 – 18
Basic Table Tags . . . . . 2 – 18
Basic Table Attributes . . . . . 2 – 19
Example Table . . . . . 2 – 20
Forms . . . . . 2 – 20
Forms Syntax . . . . . 2 – 21
Form Tags . . . . . 2 – 22
Form Selection Menus . . . . . 2 – 24
Creating Your Own HTML Document . . . . . 2 – 26
More Information about HTML . . . . . 2 – 27

Chapter 3 Oracle Web Listener . . . . . 3 – 1
Oracle Web Listener Features . . . . . 3 – 2
High Performance Architecture . . . . . 3 – 2
File Caching in Memory . . . . . 3 – 2
Memory Mapping of Files . . . . . 3 – 2
Directory Mapping . . . . . 3 – 3
Language Extensions . . . . . 3 – 3
HTTP Protocol Negotiation Features . . . . . 3 – 4
Common Gateway Interface (CGI) . . . . . 3 – 4
Imagemap Support . . . . . 3 – 7
Domain Name Server (DNS) Resolution . . . . . 3 – 7
Client to Web Listener Security . . . . . 3 – 8
Logging . . . . . 3 – 9
Starting Up and Shutting Down the Oracle Web Listener . . 3 – 9
Oracle Web Listener Configuration Parameters . . . . . 3 – 10
Logging Configuration Parameters . . . . . 3 – 12
Directory Mapping Parameters . . . . . 3 – 13
File Cache Definitions . . . . . 3 – 13
Encoding Definitions . . . . . 3 – 14
MIME Types Definitions . . . . . 3 – 15
Language Extensions . . . . . 3 – 15
Basic and Digest Authentication Configuration . . . . . 3 – 17
Oracle Web Listener Miscellaneous Configuration Parameters . . . . . 3 – 19

Chapter 4 Introduction to the Oracle7 Server and to PL/SQL . . . . . 4 – 1
What is the Oracle7 Server? . . . . . 4 – 1
Database Tables . . . . . 4 – 2

MAC003045

ORCL00941721

Users, Connections, Privileges, and Roles . . . . . 4 – 4
Distributed Databases . . . . . 4 – 5
What is SQL? . . . . . 4 – 5
Retrieving Data . . . . . 4 – 6
Creating Tables . . . . . 4 – 7
Inserting and Manipulating the Data . . . . . 4 – 8
Querying Multiple Tables Through Joins . . . . . 4 – 10
Where to Look for More Information . . . . . 4 – 11
What is PL/ SQL? . . . . . 4 – 12
Basic Structure and Syntax . . . . . 4 – 13
The DECLARE Section . . . . . 4 – 14
The EXECUTABLE Section . . . . . 4 – 21
Assignments . . . . . 4 – 21
Flow Control . . . . . 4 – 21
The EXCEPTION Section . . . . . 4 – 25
Declaring Exceptions . . . . . 4 – 26
Handling Exceptions . . . . . 4 – 27
Storing Procedures and Functions in the Database . . . . . 4 – 28
Privileges Required . . . . . 4 – 29
Packages . . . . . 4 – 30
Database Triggers . . . . . 4 – 34
Creating Triggers . . . . . 4 – 35

Chapter 5 Oracle Web Agent . . . . . 5 – 1
Components of the Oracle Web Agent . . . . . 5 – 2
Who Uses the Oracle Web Agent . . . . . 5 – 2
How the Oracle Web Agent Works . . . . . 5 – 3
Web Agent Service . . . . . 5 – 5
How the Web Agent Uses CGI Environment Variables . . . . . 5 – 8
Passing parameters to PL/ SQL . . . . . 5 – 9
Overloading Procedures . . . . . 5 – 15
Accessing CGI Environment Variables . . . . . 5 – 16
Oracle Web Agent Error Handling . . . . . 5 – 17
Creating Error Pages . . . . . 5 – 18
Error Log File . . . . . 5 – 19
Oracle Web Agent Version Number . . . . . 5 – 19

Chapter 6 Oracle WebServer Developer's Toolkit . . . . . 6 – 1
Installing the Oracle WebServer Developer's Toolkit . . . . . 6 – 2
Optimizing Multiple–Service Installations . . . . . 6 – 3
Security Note . . . . . 6 – 4

MAC003046

ORCL00941722

Procedure and Function Reference . . . . . 6 – 4
Parameters Passed into Procedures and Functions . . . . . 6 – 5
Print Procedures . . . . . 6 – 5
htp.print . . . . . 6 – 6
htp.p . . . . . 6 – 6
htp.prn . . . . . 6 – 6
htp.prints . . . . . 6 – 6
htp.ps . . . . . 6 – 6
Structure Tags . . . . . 6 – 7
htp.htmlOpen . . . . . 6 – 7
htp.htmlClose . . . . . 6 – 7
htp.headOpen . . . . . 6 – 7
htp.headClose . . . . . 6 – 7
htp.bodyOpen . . . . . 6 – 8
htp.bodyClose . . . . . 6 – 8
Head Related Tags . . . . . 6 – 8
htp.title . . . . . 6 – 8
htp.htitle . . . . . 6 – 9
htp.base . . . . . 6 – 9
htp.isindex . . . . . 6 – 10
htp.linkRel . . . . . 6 – 10
htp.linkRev . . . . . 6 – 10
htp.meta . . . . . 6 – 10
Body Tags . . . . . 6 – 11
htp.line . . . . . 6 – 11
htp.hr . . . . . 6 – 11
htp.nl . . . . . 6 – 11
htp.br . . . . . 6 – 12
htp.header . . . . . 6 – 12
htp.anchor . . . . . 6 – 12
htp.mailto . . . . . 6 – 13
htp.img . . . . . 6 – 13
htp.para . . . . . 6 – 13
htp.paragraph . . . . . 6 – 14
htp.address . . . . . 6 – 14
htp.comment . . . . . 6 – 14
htp.preOpen . . . . . 6 – 14
htp.preClose . . . . . 6 – 15
htp.blockquoteOpen . . . . . 6 – 15
htp.blockquoteClose . . . . . 6 – 15
List Tags . . . . . 6 – 15
htp.listHeader . . . . . 6 – 16
htp.listItem . . . . . 6 – 16

MAC003047

ORCL00941723

htp.ulistOpen . . . . . 6 – 16
htp.ulistClose . . . . . 6 – 17
htp.olistOpen . . . . . 6 – 17
htp.olistClose . . . . . 6 – 17
htp.dlistOpen . . . . . 6 – 17
htp.dlistClose . . . . . 6 – 17
htp.dlistDef . . . . . 6 – 18
htp.dlistTerm . . . . . 6 – 18
htp.menulistOpen . . . . . 6 – 18
htp.menulistClose . . . . . 6 – 18
htp.dirlistOpen . . . . . 6 – 19
htp.dirlistClose . . . . . 6 – 19
Character Format Tags . . . . . 6 – 19
htp.cite . . . . . 6 – 20
htp.code . . . . . 6 – 20
htp.emphasis . . . . . 6 – 20
htp.em . . . . . 6 – 20
htp.keyboard . . . . . 6 – 20
htp.kbd . . . . . 6 – 21
htp.sample . . . . . 6 – 21
htp.strong . . . . . 6 – 21
htp.variable . . . . . 6 – 21
Physical Format Tags . . . . . 6 – 22
htp.bold . . . . . 6 – 22
htp.italic . . . . . 6 – 22
htp.teletype . . . . . 6 – 22
Form Tags . . . . . 6 – 23
htp.formOpen . . . . . 6 – 23
htp.formClose . . . . . 6 – 23
htp.formCheckbox . . . . . 6 – 24
htp.formHidden . . . . . 6 – 24
htp.formImage . . . . . 6 – 24
htp.formPassword . . . . . 6 – 25
htp.formRadio . . . . . 6 – 25
htp.formReset . . . . . 6 – 25
htp.formSubmit . . . . . 6 – 26
htp.formText . . . . . 6 – 26
htp.formSelectOpen . . . . . 6 – 26
htp.formSelectOption . . . . . 6 – 27
htp.formSelectClose . . . . . 6 – 28
htp.formTextarea . . . . . 6 – 28
htp.formTextareaOpen . . . . . 6 – 28
htp.formTextareaClose . . . . . 6 – 29

Table Tags . . . . . 6 – 29
htp.tableOpen . . . . . 6 – 29
htp.tableClose . . . . . 6 – 29
htp.tableCaption . . . . . 6 – 30
htp.tableRowOpen . . . . . 6 – 30
htp.tableRowClose . . . . . 6 – 30
htp.tableHeader . . . . . 6 – 30
htp.tableData . . . . . 6 – 31
OWA_UTIL Package . . . . . 6 – 31
owa_util.signature . . . . . 6 – 31
owa_util.signature (cname) . . . . . 6 – 32
owa_util.showsource (cname) . . . . . 6 – 32
owa_util.showpage . . . . . 6 – 32
owa_util.get_cgi_env(function) . . . . . 6 – 33
owa_util.print_cgi_env . . . . . 6 – 33
owa_util.mime_header . . . . . 6 – 34
owa_util.get_owa_service_path (Function) . . . . . 6 – 34
owa_util.tableprint . . . . . 6 – 34
Customized Extensions to HTP Packages . . . . . 6 – 37

Chapter 7
The Oracle WebServer Administration Utility . . . . . 7 – 1
Database Administration . . . . . 7 – 2
Startup Modes . . . . . 7 – 3
Shutdown Modes . . . . . 7 – 3
Web Agent Administration . . . . . 7 – 4
Creating or Modifying Web Agent Services . . . . . 7 – 4
The Oracle Web Agent Administration Service Creation Form . . . . . 7 – 5
The Oracle Web Agent Administration Service Modification Form . . . . . 7 – 9
Securing a Web Agent Service . . . . . 7 – 10
Web Listener Administration . . . . . 7 – 11
The Oracle Web Listener Creation Form . . . . . 7 – 12
The Configure Extended Parameters Form . . . . . 7 – 16
Configure Security Form . . . . . 7 – 25

Appendix A
Oracle WebServer Messages . . . . . A – 1
00001 – 00600 Generic Oracle WebServer Configuration Messages . . . . . A – 2
05000 – 05499 Oracle Web Agent Messages . . . . . A – 3
05500 – 05599 Oracle Web Agent Administration Messages . . . . A – 5
05600 – 05699 Oracle Web Database Administration Messages . A – 7

MAC003049

ORCL00941725

05700 – 05799 Oracle Web Listener Configuration Messages . . . A – 8
05900 – 05999 Oracle Web Server Configuration Messages . . . . . A – 9

Index

MAC003050

ORCL00941726

CHAPTER

1 Overview

This chapter explains the concepts used in the Oracle WebServer to enable the creation of Web applications using the full power of the Oracle7 Server. The Oracle WebServer provides an integrated product that facilitates the creation of Web sites that are robust, interesting, easy to use, and provides a way for Web–based users to interact securely with data stored in the Oracle7 Server, the world's most popular relational database.

MAC003051

ORCL00941727

What is the Oracle WebServer?

The Oracle WebServer is an HTTP server with a tightly integrated Oracle7 Server that enables the creation of dynamic HTML documents from data stored in an Oracle database. When the data changes, these HTML documents are updated automatically, with no further effort on the part of the site administrator. This approach supplements the presentation of static, or unchanging, data which is found on most Web sites today, with the dynamic real–time data present in business systems based on the Oracle7 Server

Business data is stored on the Oracle7 Server. It is formatted into Web documents within the server and then transmitted to Web clients. All data is stored only once, eliminating the need to "snapshot" data periodically for use on the Web.

The components that make up the Oracle WebServer work together to make the delivery of both static and dynamic pages possible.

Components of the Oracle WebServer

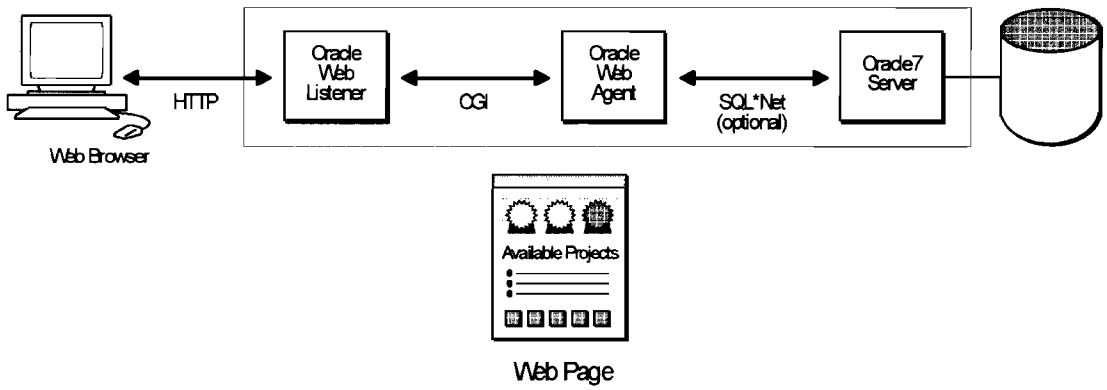
The components of the Oracle WebServer are as follows:

- Oracle Web Listener. The Oracle Web Listener receives requests from users using any Web browser. Requests for static (file–based) pages are handled internally by the Listener process, which functions as an HTTP server.
- Oracle Web Agent. The Oracle Web Agent handles requests from users for dynamic (program–created) pages. It handles the connection to an Oracle7 Server, invocation of the requested procedure, and transmission of the resulting HTML document back to the Web browser.
- Oracle WebServer Developer's Toolkit. The Oracle WebServer Developer's Toolkit is a set of procedures that help the customer create programs to generate dynamic documents that are easily presented in HTML.
- Oracle7 Server. The Oracle7 Server provides the storage for all dynamic data in relational tables, and all the program logic used to create dynamic HTML pages.

Each of these components and their interaction are described in more detail in the sections that follow.

The following picture illustrates how the Oracle WebServer integrates its components, bringing the power of Oracle7 to the World Wide Web.

http://www.oracle.com/invservice/owa/oracle/home



The Oracle Web Listener

The Oracle Web Listener is a commercial quality HTTP server that services document requests from any Web browser. Clients can be located anywhere on a network using the standard HTTP protocol. Under this protocol, the type of the document is sent to the client along with the document itself. The Web Listener supports many different document types are supported. Clients either interpret and display the document appropriately, or pass the document to a program that specifically handles that document type. For example, an MPEG viewer on the client machine may receive a video clip drawn from a newscast.

The Oracle Web Listener can handle a large number of simultaneous requests, and has advanced features which use system resources more efficiently than other HTTP servers available on the market. These features are described in more detail in Chapter 3, "Oracle Web Listener".

When the Oracle Web Listener receives a request from a client, it first determines whether that request is for a static document or a dynamic document. If the request is for a static document, the Web Listener sends the file and the associated type information directly to the client. If the request is for a dynamic document, it is created "on the fly" by a program invoked by the Web Listener, in compliance with the Common Gateway Interface (CGI). CGI is an interface that enables HTTP servers to run a program and use the output of that program in a document that is sent to the user. Most HTTP servers support this interface, and the Oracle Web Listener uses it to invoke the Oracle Web Agent when a

database procedure is requested. In addition, any other executable program may be invoked through CGI.

For further information about the Web Listener, see Chapter 3.

The Oracle Web Agent

The Oracle Web Agent is a program that is invoked by the Oracle Web Listener when a request for a database procedure is received. It handles the details of making a connection to the Oracle7 Server. A Web Agent will connect to a single Oracle7 Server, using a specific database username and password which are specified as part of a Web Agent service. To connect to different servers, or different schemas on the same server, multiple Web Agent services may be configured on a single Oracle WebServer. This allows for great flexibility in the creation of applications that unify data from several different servers, while controlling exactly what information a Web client can access.

For more information on the Oracle Web Agent, please see Chapter 5, "Oracle Web Agent".

The Oracle WebServer Developer's Toolkit

The Oracle WebServer Developer's Toolkit is a set of PL/ SQL routines and sample applications that help the user quickly create a Web application using the Oracle WebServer. The samples can serve as templates for user applications. With some simple modifications to include text and graphics specific to the user's site, these samples can be quickly and easily customized for a particular site.

When the Oracle Web Agent logs into the Oracle7 Server, it starts a PL/ SQL procedure that has been created by the user to generate an HTML page as its output. The procedures and functions in the Developer's Toolkit generate the actual HTML tags, so that the user can be focused on the content of the application, rather than the specifics of HTML.

For detailed information on the routines included in the Oracle WebServer Developer's Toolkit, see Chapter 6 "Oracle WebServer Developer's Toolkit".

MAC003054

ORCL00941730

Oracle7 Server

The Oracle7 Server is a high performance, fault–tolerant relational database management system, especially designed for on–line transaction processing and large database applications. The Oracle7 Server reliably manages a large amount of data in a multi–user environment. The Oracle7 Server delivers high performance even while many users concurrently access the data. It is also secure from unauthorized access and provides efficient solutions for failure recovery.

For a discussion of the Oracle7 Server, see Chapter 4 in this manual and the documentation listed in the Preface of this book.

Summary

The integrated package provided by the Oracle WebServer allows the creation of dynamic applications that take advantage of the power of Oracle7 database technology and the interactive appeal of the World Wide Web. In the following chapters, each component of the product is described in more detail.

MAC003055

ORCL00941731

MAC003056

ORCL00941732

CHAPTER

2 Introduction To HTML

This chapter discusses the basic concepts of HyperText Markup Language, or HTML. This chapter is an introduction to HTML and provides important information on how to set up, format, and define HTML documents.

The chapter covers the following topics:

- Basic concepts
- Getting started
- HTML tags
- Hypertext Linking
- Graphics
- Tables
- Forms
- Creating your own document
- On–line information about HTML

Note: This chapter contains examples of formatted elements, such as underlined text. Some of these examples may not appear correctly on all on–line viewing systems.

MAC003057

ORCL00941733

What is HTML?

Hypertext Markup Language (HTML) is the standard language used for creating hypermedia documents on the Web. HTML documents can be viewed by many different Web browsers, of varying abilities, in a simple, portable way. When a document is coded in HTML, a browser. can interpret the HTML to identify the elements of the document and to render it. The use of HTML allows documents to be formatted for presentation using fonts and line justification appropriate for the system on which it is displayed.

Most documents have common elements, such as a title, paragraphs, or lists. Using HTML tags you can label these elements as you are writing. HTML tags provide the browser with a minimum of presentation information, while keeping the integrity of information in the document. All the reader needs is a formatting tool, a Web browser, which interprets the HTML tags and produces an on–screen display that approximates the intent of the document creator.

With most methods of documentation, the writer of a document has strict control over the look and feel of a document. With HTML, the reader (subject to the capabilities of the Web browser) has control over the look and feel of a document. HTML allows you to mark titles or paragraphs with HTML instructions or tags, and then leaves the interpretation of these tags up to the browser. For example, one browser may indent the beginning of each paragraph, and another may leave only a blank line. The user of a particular browser may also have some control over the specific fonts used.

HTML tags can be divided into two main categories:

- tags that define how the body of the document is to be displayed by the browser
- tags that define information about the document such as the title

Remember, the power of HTML is that your document can be viewed on a variety of browsers, on most platforms, and formatted to suit any reader.

How Are HTML Documents Created?

HTML documents can be created using any text editor (emacs, textedit, or vi on UNIX machines; DOS and Macintosh machines have a variety of simple text editors or HTML–specific programs). Choose the editor you are most comfortable with to write your HTML document.

MAC003058

ORCL00941734

For example, HTML editors such as SoftQuad's "HoTMetaL" allow the creation of HTML documents graphically in "what you see is what you get" (WYSIWYG) mode. In addition, many traditional word processing packages have add–ons or integrated HTML output capabilities.

To create dynamic pages that retrieve information from an Oracle7 Server, you can generate HTML with the PL/ SQL utility packages provided by the Oracle WebServer Developer's Toolkit. See Chapter 5, "Oracle Web Agent," and Chapter 6, "Oracle WebServer Developer's Toolkit," for more information.

Getting Started

All HTML tags begin with a < (left angle bracket), and end with a > (right angle bracket). There is usually a beginning tag and an ending tag.

An example is the title tag which surrounds the text that is designated as the document's title:

```
<TITLE>All the Hockey Greats</TITLE>
```

Tags are usually paired as follows:

```
<TITLE> and </TITLE>
```

The ending tag looks like the beginning tag except that a forward slash precedes the text within the bracket. In this example, the tag <TITLE> tells the Web browser to use a title format, and the </ TITLE> tells the browser that the title heading is complete.

A few tags, such as <P>, which is a paragraph delimiter, do not need an end tag, but most do.

HTML is not case sensitive; therefore, the previous tags could look like this:

```
<title>All the Hockey Greats</title>
```

The convention used in this document is to capitalize all HTML format tags.

Note: Extra spaces, tabs, or returns that you have added by hand are highly discouraged, and will be lost. HTML only interprets tabs, extra spaces, and returns enclosed by the <PRE> </ PRE> tags. See "Preformatted Tag" for more information on this HTML option.

Document Structure

When a browser receives a document, it determines how it should be interpreted. The very first tag you need in your HTML document is the <HTML> structure tag. This declares that the content of your document is written with HTML. A minimal HTML document would look like this:

```
<HTML>. . . t he cont ent of t he document . . .
</ HTML>
```

Head Tag

The head tag can be used right after the HTML declaration, or not at all in your document. This tag represents the prologue to the rest of the file. Avoid putting any text into the document <HEAD> tag. This tag is placed immediately before and after the <TITLE> tag, as shown in the following example:

```
<HTML>
<HEAD>
<TI TLE>Al l t he Hockey Gr eat s</ TI TLE>
</ HEAD>
```

Note: Technically, the start and end tags for <HTML>, <HEAD>, and <BODY> are not needed. However, they are recommended because the head and body structure tags allow a browser to determine certain properties of a document, such as the <TITLE>, without having to parse, or go through the whole document.

Title Tag

Most browsers display the contents of the <TITLE> tag in the title bar of the window containing the document, and in the bookmark file of the browser if it supports one. The title, surrounded by <TITLE> and </ TITLE> tags, is placed between the Head tags, as shown above. The title of a document does not appear in the contents of the document window, however. You must separately indicate it as a heading inside the body of the document if you want it to appear there.

Body Tags

The body tags specifically identify the body components in an HTML document. The body of an HTML document can contain links, text, and formatting information inside of the <BODY> and </ BODY> tags.

MAC003060

ORCL00941736

Body Tag

The body of the document should be marked off with the <BODY> and </ BODY> tags. This is the part of the document that is displayed as the page of text and graphics on your Web browser.

Heading Levels

When writing an HTML document, organize the text by heading levels to reflect its structure and organization. The first heading would be level 1, the next sub heading level 2, and so on. Most browsers recognize up to six heading levels, with six distinct styles. Heading levels above 6 are *indistinguishable from one another.*

The largest heading is a level 1 heading. The syntax of the head 1 is as follows:

```
<H1>Hockey Greats on Offense</H1>
```

Other headings can be created as follows:

```
<Hx>Text here</Hx>
```

where x is a number between 1 and 6 specifying the heading level. For example, if your next heading level is a level 3, the syntax would look like the following:

```
<H3>Hockey Defense </H3>
```

Paragraph Tag

Unlike most word processors, HTML usually ignores carriage returns. Word wrapping can occur at any point in your source file. Therefore, paragraphs must be separated with the <P> tag. If you do not separate your paragraphs with the <P> tag, your document will look like one long paragraph.

Preformatted Tag

The preformatted tag, <PRE>, allows you to present text formatted specifically to a screen. The preformatted text ends at the closing </ PRE> tag. Within the preformatted text:

- Line breaks move to the next line
- The <P> tag is moved to the next line
- Horizontal tabs move in multiples of eight
- A fixed width font is used for all characters after the <PRE> tag.

Avoid using tags that define paragraph formatting, such as headings or address, within the <PRE> tags. They will have no effect.

Let's incorporate some of the previous examples to show what the document looks like with a title, a couple of heading level tags, and a few paragraphs:

```
<HTML>
<HEAD>
<TITLE>All the Hockey Greats</TITLE>
</HEAD>
<BODY>
<H1>All the Hockey Greats Before 1970</H1>
<H2>The Original Six Teams</H2>
This section deals with all of the hockey legends before
the expansion.<P>
The game was very different for these players.<P>
There is no way New Jersey would have won the Stanley Cup.
New Jersey just would not have had the talent to do it.<P>
Chicago would still be on top.<P>
All would be well.<P>
</BODY>
</HTML>
```

The result would display something like this:

All The Hockey Greats Before 1970

The Original Six Teams

This section deals with all of the hockey legends before the expansion.

The game was very different for these players.

There is no way New Jersey would have won the Stanley Cup. New Jersey just would not have had the talent to do it.

Chicago would still be on top.

All would be well.

Note: The title, "All the Hockey Greats," would not show within the document itself. On most browsers it would be displayed in the title bar.

Forced Line Breaks

The <BR> tag forces a line to break . The best example of the use of this tag is for formatting addresses, or some other sequence of lines where you don't want the browser to add extra spacing. For example:

MAC003062

ORCL00941738

```
Sandy's Super Sundaes<BR>
123 Main Street<BR>
Anytown, USA<BR>
```

BlockQuote

The <BLOCKQUOTE> tag is used to contain text quoted from another source. The quote will be indented approximately 8 spaces. For example:

```
My favorite hockey saying is<P>
<BLOCKQUOTE>
Today is a great day for hockey.
</BLOCKQUOTE>
But I'm not sure if Bob Johnson really said it like
that.</P>
```

This would appear something like the following:

My favorite hockey saying is

> Today is a great day for hockey.

But I'm not sure if Bob Johnson really said it like that.

Summary of Basic HTML Tags

The following table lists the basic HTML tags and their corresponding values:

Opening	Closing	Definition
<HTML>	</HTML>	An entire HTML document
<HEAD>	</HEAD>	The prologue of the document
<TITLE>	</TITLE>	Title of the document
<BODY>	</BODY>	Content of the document
<H1>	</H2>	First level heading
<H2>	</H2>	Second level heading
<H3>	</H3>	Third level heading
<H4>	</H4>	Fourth level heading
<H5>	</H5>	Fifth level heading
<H6>	</H6>	Sixth level heading

MAC003063

ORCL00941739

Opening	Closing	Definition
<P>		Paragraph
<PRE>	</ PRE>	Preformatted text
 		Forced line break
<BLOCKQUOTE>	</ BLOCKQUOTE>	Text quoted from another source

The previous section provides all you need to know to get started with HTML. At this point you can write a simple document in HTML. However, the following sections show you how to enhance your HTML pages to present information in many different fashions.

List Tags

There are three basic lists in HTML:

ordered These lists have numbered items.

unordered These lists have bullets to mark each item.

definition These lists alternate a term with its definition

You can create nested lists with indents using the ordered or unordered tags. Simply place a second list (complete with its own start and end tags) within the first lists enclosing tags. Whether the nested list uses the same markers (numbers or bullets,) depends on the browser; some track the number of nests you use and change the markers of each successive nesting to blocks or other symbols. See the following example in the section "Nested Lists".

Ordered Lists

In an ordered list, the browser automatically inserts numbers. Therefore, if you insert or delete an item in your ordered list, the numbers will reflect the change automatically.

An ordered list begins with <OL> and ends with </ OL>. The individual list items are started with the <LI> tag. The following is an example of an ordered list:

```
<OL>
<LI>Gordie Howe
<LI>Rocket Richard
```


MAC003064

ORCL00941740

```
<LI>Howie Morenz
</OL>
```

Unordered Lists

In an unordered list the browser typically uses bullets or dashes to indicate the items in your list. (Each browser has its own way of indicating an unordered list)

An unordered list begins with <UL> and ends with </UL>. The following is an example of an unordered list:

```
<UL>
<LI>Gordie Howe
<LI>Rocket Richard
<LI>Howie Morenz
</UL>
```

Nested Lists

Here is an example of how to nest a list within another:

```
<HTML>
<HEAD>
<TITLE>All the Hockey Greats</TITLE>
</HEAD>
<BODY>
<H1>Hockey Greats before Expansion</H1>
<H2>The Original Six Teams</H2>
This section deals with all of the hockey legends before
the expansion. <P>
<UL>
<LI>Gordie Howe
<LI>Rocket Richard
<LI>Howie Morenz
<UL>
<LI>great player
<LI>good stickhandler
</UL>
<LI>Bobby Orr
</UL>
</BODY>
</HTML>
```

Here's what the example would look like:

Hockey Greats before Expansion

The Original Six Teams

This section deals with all of the hockey legends before the expansion.

- Gordie Howe
- Rocket Richard
- Howie Morenz
  - great player
  - good stickhandler
- Bobby Orr

Note: When you create nested lists using HTML, you do not need to indent the nested HTML components. You may wish to do so for clarity, however.

Definition Lists

The definition list tags <DL>...<D/ DL> enclose both the defined term (identified with the <DT> tag), and the definition of that term (identified with the <DD> tag). Most browsers format the definition on a separate line from the term. The following is an example of a definition list:

```
<DL>
<DT>Slapshot:
<DD>A shot used to drill the goalie at speeds up to 100
mph.
<DT>Wrist shot:
<DD>A shot used to scare the goalie after the slapshot.
</DL>
```

The output looks like this:

Slapshot:
A shot used to drill the goalie at speeds up to 100 mph.

Wristshot:
A shot used to scare the goalie after the slapshot.

MAC003066

ORCL00941742

Hypertext Linking

Hypertext linking is the key characteristic that makes the Web appealing to users. By adding hypertext links, called anchors in your HTML document, you can create a highly intuitive information flow and guide the user directly to the information he or she needs.

Anchors have a standard format that allows any Web browser to interpret a link and perform the proper function (called a method) for that type of link. Links can refer to other documents, specific locations within the same document, or can perform operations, such as retrieving a file using FTP for display by the browser. URLs can refer to a specific location by an absolute pathname, or can be relative to the current document, which is more convenient when managing a large site.

> Note: You can use hypertext links to navigate through a document or to move from document to document. However, HTML does not support returning you to the anchor point of a link within a document. If you use a hypertext link within a document, and then use the Back button, you do not return to the anchor, but to the previous point that you reached through a link.

What is a URL?

HTML uses what are called Uniform Resource Locators (URLs) to represent hypermedia links and links to network services within documents. The first part of the URL before the colon specifies the access method. The part of the URL after the colon is interpreted specifically according to the access method. In general two forward slashes after the colon indicate a machine name.

The general format of a URL is:

method://machine-name/*path/foo.html*

The following example would fetch the document index.html from the server www.acme.com using the HTTP protocol.

http://www.acme.com/index.html

A Uniform Resource Locator (URL) has the following format:

method://servername:port/pathname#anchor

The components of the URL are as follows:

method — is the name of the operation that is performed to interpret this URL. The most common methods are:

file	Read a file from the local disk. The filename is interpreted on the user's machine, so this can be used to display any file that resides on the user's disk. For example: `file:/home/jjones/jjones.html` displays the file `jjones.html` from the directory `/home/jjones` on the local machine.
http	Access a page over the network by way of the HTTP protocol. (This is the most common method, usually used to get an HTML document.) For example: `http://www.acme.com/` accesses Acme's home page.
mailto	Activate a mail session to the specified username and host. For example: `mailto:jjones@us.acme.com` mails a message to jjones if the browser supports mail creation. Note that the `mailto` method does not require double forward slashes after the colon.
ftp	Retrieve a file using anonymous FTP from a server. For example, `ftp://`*`hostname/directory/filename`
servername	is optional and indicates the full hostname of a machine connected to the network. For example, `www.oracle.com` is the fully qualified hostname of Oracle's web server. If a servername is not specified, the URL is a relative link, and it is assumed that the file is on the same server that was used to display the current page. An IP address may be used instead of a hostname, although it is not recommended to build content with embedded IP addresses.
port	is the TCP port number that the web server is running on. The default is 80 if `:port` is not specified. This parameter is not used in most URLs.
pathname	is the relative or absolute pathname of the document being accessed by this URL. Web servers can be configured to interpret certain pathnames differently. For instance, CGI applications work by configuring the HTTP server to recognize that files within certain directories should be executed instead of being returned to the browser. For example: `http://www.acme.com/index.html`

MAC003068

ORCL00941744

In this example, an HTTP connection is made to index.html, which is the name of the file to be accessed on the server www.acme.comusing port 80 (the default). The file could have a full UNIX–style pathname to indicate a document contained in a lower level directory. If there is no directory component in the pathname, the document must be located in the server's document root directory which is configured by the server administrator. If the pathname part of the URL is missing, many servers provide a directory listing of the document root directory or access a specific 'top level' file (usually index.html).

#anchor — The named anchor points to a specific location within an HTML page. In addition to specifying a document name, specifying #anchor will cause most browsers to move the top of the display to the point referred to by the anchor. These anchor names are inserted into documents with the NAME tag as explained under "Linking to Sections on Pages" later in this chapter.

Structure of an Anchor Link within a Document

So far, we've discussed what a URL looks like. To cause a link to be displayed for the user to access, an anchor link must be embedded in the document text. The HTML syntax which allows this is:

<A HREF="*URL*">*text_to_be_displayed_highlighted*</A>

The <A HREF="URL"> tag opens the anchor link and the </A> tag closes it. All the text between those tags is displayed highlighted in some way by a web browser. A common technique is to display it underlined and in blue or some other user–selected contrasting color.

The URL part of the tag refers to the text of the URL reference as defined in the previous section. The URL text does not appear on the user's screen; it is only used when the user activates the link, usually by clicking on it with the mouse.

An example HTML segment:

```
For interesting products see
<A HREF="http://www.acme.com">Acme's home page.</A>
```

This line would produce on the user's screen:

For interesting products see Acme's home page.

Graphics within HTML Documents

One of the most compelling features of the Web is the ability to embed references to graphics and other data types within a document using the <IMG...ISMAP> tags. This adds a very lively character to your pages and makes them visually interesting.

There are two ways to use graphics from within an HTML document. The first is by embedding them within the document itself, so the user's screen will display the graphics within the context of the other elements of your document (such as explanatory text). This is the most common technique used by HTML designers and is called an "inline image". The syntax for specifying this is:

```
<IMG SRC="URL" ALT="text"
    ALIGN=["top" | "middle" | "bottom" | "texttop" ] ISMAP>
```

The elements in this syntax statement are as follows:

URL	is the same syntax as any other URL, as explained above. This is the way the browser accesses the actual image data file, which should be in a format supported by the browser. Currently GIF and JPEG formats are supported by most browsers. Specifying the URL is required.
ALT="text"	will cause the string text to be displayed if the browser is incapable of displaying images, or if image display is turned off. This is a way of 'labelling' the image the user would be seeing if image display was turned on. ALT is an optional keyword; if it is missing, no text will be displayed if images are turned off on the browser. Most browsers put some sort of icon on the screen to indicate an image would normally be there. Using the ALT tag is recommended so that your page is compatible with text–only browsers such as Lynx.
ALIGN	is used with one of the keywords to tell the browser where to place the next block of text. This allows a certain amount of creativity in the layout of your page. If this is not specified, most browsers put the image on the left side of the screen and fill in following text to the right of it.
ISMAP	tells the server that this image is a bitmap and allows the user to click on a location on the image to cause a URL to be accessed directly. Image maps are an advanced HTML feature and require server side configuration to function.

MAC003070

ORCL00941746

For example, the following line will cause the file logo.gif to be fetched from server www.oracle.com and the text Oracle Logo to be displayed if the user has graphic display turned off.

```
<IMG SRC="http://www.oracle.com/logo.gif" ALT="Oracle Logo">
```

Linking to Sections on Pages

You can link to a different area or section of your document by using a hidden reference marker to that specific section. This provides a quick way to move through sections of the document without having to scroll up or down. Once you click on that link, the hidden reference places you in the section, and the Browser presents the hidden marker line as the first line on the screen.

To create a link, follow these steps:

1. Create a named anchor marker in each section title that you might want to jump to. The HTML syntax is as follows:

   ```
   <A NAME="named_anchor">Text_to_link_to</A>
   ```

   In the following example, a hidden reference marker is placed in a Head2 level:

   ```
   <H2><A NAME="intro">Introduction to Hockey</A></H2>
   ```

2. Create the link by entering:

   ```
   <A HREF="#named_anchor">Text</A>
   ```

   For example, the link might appear as:

   ```
   <A HREF="#intro">Gordie Howe</A>
   ```

The "#" symbol instructs your web browser to look through the HTML document for a named anchor called "intro".

When the user clicks on "Gordie Howe," the browser displays the heading "Introduction to Hockey" at the top of the screen.

Note: As specified earlier in the URL syntax, a link to a section can appear in the same document or in another document. The example here describes a link to a specific section of the same document.

Reviewing Changes to Your HTML Document

At some point, you will want to view the changes you have made to your HTML document. The following steps show you how to view your changes:

1. After you have edited your HTML file in your favorite text editor, save the file.
2. If you are currently displaying the document in a browser, you will need to reload it to see the changes. Select File —>Reload on the browser window. (In some browsers, reload will be under another menu item.)

The browser will read in the new file information and display the file with the changes you have made.

Adding Style to Your HTML Document

HTML tags offer several text styles so that you can emphasize any text in your document. The following is a short list of the most often used tag styles :

- bold
- italics
- underline
- mono spaced (typewriter style)

You can also use combinations of styles (for example, bold and italics)

Style	Element or Tag	Result
Bold	<B>I want this text bold</ B>	I want this text bold
Italics	<I>I want this text italics</ I>	I want this text italics
Underlined	<U>I want this under-lined</ U>	I want this underlined
Mono spaced(typewriter)	<TT>I want this text typed</ TT>	I want this text typed

You can add style to text that appears anywhere in the HTML document. Combinations of styles can also be used, provided you assign all the closing tags needed.

<I >Hockey</ I > i s <B>Li f e</ B>. <P>

becomes a paragraph that looks like this:

Hockey is Life.

The style tags surround the words they affect, in conjunction with other tags such as headings.

Be careful how many style tags you use on one page. If you use too many, the text may become difficult to read.

Special HTML Tags

The following tags or characters are options provided to make your HTML document more robust.

Address Tag

The <ADDRESS> tag is used to specify the author of a particular HTML document, and a way to contact the author (such as an email address). The syntax is as follows:

```
<ADDRESS>
address_of_author
</ ADDRESS>
```

Escape Sequences

The following ASCII characters have special meaning within HTML and cannot be used in regular text:

- left angle bracket <
- right angle bracket >
- ampersand &
- double quote "

To use these characters you must use these escape sequences:

For	Use
<	<
>	>
&	&
"	"

There are several more escape sequences to support accented characters, such as the umlaut or the tilde. A full list of supported characters can be

MAC003073

ORCL00941749

found in any number of reference manuals on HTML syntax. See also the list of on–line HTML references at the end of this chapter.

Tables

Tables in HTML organize data by row and column. Tables can contain a wide range of content, such as headers, lists, paragraphs, or figures. They can include any element or tag in HTML.

Cells can be merged across rows or columns.

Basic Table Tags

This section describes the basic table tags and their meanings.

Table: <TABLE>. . . </ TABLE>

This is the main wrapper for all the other table tags. Other table tags can be ignored if they aren't wrapped inside of the <TABLE></ TABLE> tags. By default, tables have no borders. Borders are added if the BORDER attribute is specified. See the next section "Basic Table Attributes".

Table Row: <TR> </ TR>

The number of rows in a table is specified by how many <TR> tags are contained within it. <TR> can have both the ALIGN and VALIGN attributes, which if specified become the default alignments for all cells in this row. See the next section, "Basic Table Attributes".

Table Data: <TD> </ TD>

This specifies a standard table data cell. Table data cells must only appear within table rows. Each row need not have the same number of cells specified, because short rows will be padded with blank cells on the right. A cell can contain any of the HTML tags normally present in the body of an HTML document. The default alignment of table data is ALIGN=left and VALIGN=middle. These alignments can be overridden by any alignments specified in the containing <TR> tag.

Note: Row alignments are overridden by any attributes assigned to a cell.

By default, lines inside of table cells can be broken up to fit within the overall cell width. Use the NOWRAP attribute described in the section, "Basic Table Attributes," to prevent line breaking for that cell.

Table Header: <TH> </ TH>

MAC003074

ORCL00941750

The table header cells are identical to data cells in all respects, except that header cells are in a bold font and have a default ALIGN=center.

Caption: <CAPTION>. . . </CAPTION>

This optional tag represents the caption for a table. The <CAPTION> tags should appear inside the <TABLE></TABLE> tags but not inside table rows or cells. The default alignment for the <CAPTION> tag is ALIGN=top but can be explicitly set to ALIGN=bottom. Like table cells, any document body HTML tags can appear in a caption. Captions are always horizontally centered with respect to the table, and they may have their lines broken to fit within the width of the table.

Basic Table Attributes

This section lists the basic table attributes and their meanings.

BORDER: This attribute appears in the Table tag. If present, borders are drawn around all table cells. If absent, there are no borders. By default space is left for borders, so a table has the same width with or without the border attribute.

ALIGN: If the ALIGN attribute appears inside a <CAPTION> </CAPTION> tag, it controls whether the caption appears above or below the table. It can have the values top or bottom. The default attribute is ALIGN=top.

When appearing inside a <TR>, <TH>, or <TD> tag, ALIGN controls whether text inside the table cell(s) is aligned to the left side of the cell, the right side of the cell, or centered within the cell. Values are left, center, and right.

VALIGN: The VALIGN attribute appears inside a <TR>, <TH>, or <TD> tag. This attribute controls whether text inside the table cell(s) is aligned to the top of the cell, the bottom of the cell, or vertically centered within the cell. It can also specify that all the cells in the row should be vertically aligned to the same baseline. Values are top, middle, bottom, and baseline.

NOWRAP: If the NOWRAP attribute appears in any table cell, the lines within this cell cannot be broken to fit the width of the cell. Be cautious in use of this attribute as it can result in excessively wide cells.

COLSPAN: The COLSPAN attribute can appear in any table cell and specifies how many columns of the table a specified cell should span. The default COLSPAN for any cell is 1.

ROWSPAN: The ROWSPAN attribute can appear in any table cell and specifies how many rows of the table this cell should span. The default

ROWSPAN for any cell is 1. A span that extends into rows that were not specified with a <TR> tag will be truncated.

COLSPEC: The COLSPEC attribute can be used when needed to exert control over column widths, either by setting explicit widths or by specifying relative widths. Specify the table width explicitly or as a fraction of the current margins.

Example Table

The following is a table using some of the table tags and attributes we have previously discussed:

```
<TABLE BORDER>
<CAPTION ALIGN=bottom> Table #1 </CAPTION>
<TR><TD ROWSPAN=2> </TD> <TH COLSPAN=2>Average</TH></TR>
<TR><TH>Height </TH><TH>Weight </TH></TR>
<TR><TD>Males</TD><TD ALIGN=center> 69 </TD>
<TD ALIGN=center>150 </TD></TR>
<TR><TD>Femal es</TD><TD ALIGN=center> 64 </TD>
<TD ALIGN=center>130 </TD></TR>
</TABLE>
```

The table will look like this:

	Average	
	Height	Weight
Males	69	150
Females	64	130

Table #1

Forms

HTML pages can be formatted in any fashion, but remain read–only. The HTML form feature brings the added advantage of being interactive. An HTML form lets the Web user enter comments and specify database search criteria.

When a form is interpreted by a Web browser, a special graphical user interface (GUI) screen is created with text entry fields, buttons, checkboxes, pull–down menus, and scrolling lists. When the Web user fills out the form and presses a button indicating the form should be

MAC003076

ORCL00941752

submitted, the information on the form is sent to an HTTP server for processing by a CGI (Common Gateway Interface) program. For more information on CGI, see Chapter 3, "The Oracle Web Listener".

When you write a form, each of your input items has an <INPUT> tag. When the user places data in these items in the form, that information is encoded into the form data and is known as the "value".

All form elements have Name and Value attributes. Data are sent as NAME=VALUE pairs, separated by ampersands (&), where name is given in the NAME attribute, and value is given in the VALUE attribute, or replaced by the user.

This section illustrates the basic use of HTML forms. Forms can be used for simple table searches or complex queries to relational databases.

Forms Syntax

All forms begin with <FORM> and end with </ FORM>.

The syntax is as follows:

<FORM METHOD="get | post" ACTION="*URL*">*Form_elements_and_other_HTML*
</ FORM>

METHOD

The request method supplies the data to the program. There are two request methods that can be used to access your forms. Depending on which request method you use, you will receive the encoded results of the form in a different way.

- GET: Information from a form is appended onto the end of the URL being requested. Your CGI program receives the encoded form input in the environment variable QUERY_STRING. Use of the GET method is discouraged.
- POST: This request method transmits all form input information immediately after the requested URL. Your CGI program will receive the encoded form input on standard input. The server will not send you an EOF on the end of the data; instead use the environment variable CONTENT_LENGTH to determine how much data you should read from standard input. This is the preferred method.

ACTION

ACTION specifies the URL being requested from the form. This URL will almost always point to a CGI script to decode the form results. If you are referring to a script on the same server as the form, you can use a relative URL.

TEXTAREA

The <TEXTAREA> tag is used to allow a user to enter more than one line of text in a form. The following is an example of the TEXTAREA tag:

```
<TEXTAREA NAME="address" ROWS=14 COLS=60>
Chicago Blackhawks
1800 Madison Ave
Chicago, Il 60612
</TEXTAREA>
```

The attributes included within the <TEXTAREA> tag are used to initialize the field's value. The </TEXTAREA> tag is always required even if the field is initially blank.

The following are attributes of <TEXTAREA> and determine the visible dimensions of the field in characters:

- NAME–user defined name
- ROWS–height in characters of TEXTAREA
- COLS–width in characters of TEXTAREA

If you want text to appear within the text area, enter it between the start and end <TEXTAREA>tags.

INPUT

The <INPUT> tag allows you to input a single word or line of text, with a default width of 20 characters. It is usually preceded with some descriptive text.

The following are attributes of <INPUT>:

- CHECKED–When present indicates that a checkbox or radio button is selected.
- MAXLENGTH–The maximum number of characters that will be accepted as input. This limits the number of characters a user can type into the field. The form will give an error beep if the user tries to enter too many characters. This can be greater that specified by SIZE, in which case the field will scroll appropriately. The default is unlimited.
- NAME–Symbolic field name used when transferring the form's contents. This attribute is always needed and should uniquely identify this field.
- SIZE–Specifies how large an area to allocate, in characters, on the screen.

- SRC–A URL specifying an image for use only with TYPE=IMAGE.
- TYPE–Defines the type of data the field accepts. Defaults to free text. Several types of fields can be defined with the TYPE attribute:

  CHECKBOX

  Used for simple Boolean attributes, or for attributes that can take multiple values at the same time. The latter is represented by a number of checkbox fields each of which has the same NAME. Each occurrence of a checkbox is either ON or OFF. For a multi–valued attribute, there is a checkbox for each attribute, indicating whether the attribute applies. The various attributes are associated with one another by the fact that each attribute checkbox uses the same name. Each selected checkbox generates a separate NAME=VALUE pair in the submitted data, even if this results in duplicate names. The default value for CHECKBOX is ON.

  HIDDEN

  No field is presented to the user, but the content of the field is sent with the submitted form. This value may be used to transmit state information about client–server interaction, or for password information.

  IMAGE

  An image field upon which you can click with a pointing device, causing the form to be immediately submitted. The coordinates of the selected point are measured in pixel units from the upper left corner of the image, and are returned (along with the other contents of the form) in two NAME=VALUE pairs. The x–coordinate is submitted under the name of the field with .x appended, and the y–coordinate is submitted under the name of *the field with .y appended. Any value attribute is ignored. The* image itself is specified by the SRC attribute, exactly as for the <IMG> tag.

  PASSWORD

  Same as TEXT attribute, except that password text is not displayed as it is entered.

  RADIO

  For attributes which can take a single value from a set of alternatives. Each radio button field in the group should be given the same NAME. Only the selected radio button in the group generates a NAME=VALUE pair in the submitted data. Radio buttons require an explicit VALUE attribute.

RESET

This is a button that when pressed resets the form's fields to their specified initial values.

SUBMIT

This button, when pressed, submits the form. You can use the VALUE attribute to provide a non–editable label to be displayed on the button. The default label is application–specific. The NAME attribute, if used, passes a name=value pair along with the submitted form.

TEXT

Single line text entry fields. Use in conjunction with the SIZE and MAXLENGTH attributes. Use the TEXTAREA tag for text fields that can accept multiple lines.

- VALUE–Assigns an initial default value for the field, or the value when checked for checkboxes and radio buttons. This attribute is required for radio buttons.

Form Selection Menus

There are three types of selection menu tag for forms:

- Select: user selects from a fixed set of values represented by the option tag. This tag is usually displayed with a pull down menu.
- Select single: same as Select, but display is presented with a window with three items displayed at once . If there are more than three options, the window will have a scroll bar.
- Select multiple: allows multiple items to be selected from the menu.

SELECT

The SELECT tag allows the user to select a value from a fixed list. This is usually presented as a pull down menu.

The SELECT tag has one or more options between the start<SELECT> and end</ SELECT> tag. By default the first option is displayed in the menu. The following is an example of a <SELECT> tag:

```
<FORM>
<SELECT NAME=group>
<OPTION>  Gretzky
<OPTION>  Messier
```


MAC003080

ORCL00941756

```
<OPTION>  Coffey
</SELECT>
</FORM>
```

SELECT SINGLE

The SELECT SINGLE tag is the same as the SELECT tag, but options are displayed in a window with three items shown at once. If there are more than three options, the window will have a scroll bar. The SIZE tag within the SELECT tag specifies how many options will be shown in the window. The following is an example of a <SELECT SINGLE>tag:

```
<FORM>
<SELECT SINGLE NAME=group      SIZE=3>
<OPTION>                Gretzky
<OPTION>                Messier
<OPTION>                Coffey
<OPTION>                Kurri
</SELECT>
</FORM>
```

In this example, the first three names would appear in the window, and a scroll bar would scroll to the last name.

SELECT MULTIPLE

The SELECT MULTIPLE tag is the same as the SELECT SINGLE tag, but the user can select more than one option in the window. The SIZE tag specifies how many lines appear in the window, and the MULTIPLE tag specifies how many options can be selected.

The following is an example of SELECT MULTIPLE:

```
<FORM>
<SELECT MULTIPLE NAME=group SIZE=3    MULTIPLE=2>
<OPTION>                Gretzky
<OPTION>                Messier
<OPTION>                Coffey
<OPTION>                Kurri
</SELECT>
</FORM>
```

Note: On some browsers, it may be necessary to hold down the CONTROL or SHIFT key to select multiple items.

If multiple items are selected, they each get passed to the server with the same name. The decoding script has to be able to recognize multiple values associated with the same name.

Creating Your Own HTML Document

Now that most of HTML has been demystified, you can create your own HTML documents. Use any browser to view the HTML document you have created.

The following is an example of an HTML document that you can type word for word, or modify as you wish:

```
<HTML>
<HEAD>
<TITLE>Chicago Blackhawks: A Love Story</TITLE>
</HEAD>
<BODY>Chicago Blackhawks: A Love Story
<P>This is a story about my beloved Blackhawks. Year after
painful year they amass teams that could potentially win a
Stanley Cup. But year after insidiously painful year, they
manage to lose in the first or second round of the
playoffs. If you have any suggestions as to how to
reconcile this evil, many fans would be grateful. Still we
live on to love the Blackhawks.
<H2>Some Players We Have Loved</H2>
<UL>
<LI><A HREF="Hull.html">Bobby Hull</A>
<LI><A HREF="Makita.html">Stan Makita</A>
<LI><A HREF="Espo.html">Tony Esposito</A>
<LI><A HREF="Jr.html">Jeremy Roenick</A>
</UL>
<H2>Some Players We Love to Hate</H2>
<OL>
<LI><A HREF="Clark.html">Wendall Clark</A>
<LI><A HREF="Gilmour.html">Doug Gilmour</A>
<LI><A HREF="Domi.html">Tie Domi</A>
<LI><A HREF="cheapshot.html">Ulf Samuellson</A>
</OL>
<H2>List of Love Letters to Our Team</H2>
<UL>
<LI><A HREF="http://www.love2hawks.com/">Love Letters</A>
<LI><A HREF="http://www.fromNHL.com/">Letters from NHL</A>
```


MAC003082

ORCL00941758

```
</UL>
<HR><A HREF="Write">Write</A>to me.
Click below to send me comments.
<BR> <A HREF="mailto:oldstadium@madison.com">
<I>crazed4hawks, oldstadium@madison.com</I>
</A>
</BODY>
</HTML>
```

More Information about HTML

For online information on HTML, the following URL sites provide a wealth of information:

```
http://www.ncsa.uiuc.edu/demoweb/html-primer.html
http://union.ncsa.uiuc.edu/HyperNews/get/www/html.html
http://fire.clarkson.edu/doc/html/htut.html
http://ugweb.cs.ualberta.ca/~gerald/guild/html.html
```

MAC003084

ORCL00941760

CHAPTER

3 Oracle Web Listener

The Oracle Web Listener is a high–performance scalable and portable HTTP server, supporting high–traffic Web applications where quick response is crucial. The Oracle Web Listener is designed to enable global distributed information systems over a corporate network or the global Internet.

From a user's perspective, the Web is a collection of documents or pages which contain text, images, and hypertext links to other pages. By pointing, clicking, and traversing the links, the user has instant access to a distributed collection of information. The Oracle Web Listener merges the techniques of information retrieval and hypertext to create a powerful global information system.

The Oracle Web Listener works on a simple client–server model. Clients send requests to the Oracle Web Listener. The listener interprets a request by reading the URL, finds (or generates) the information requested, and returns that information to the client in the form requested.

You can run several Oracle Web Listeners on the same machine by giving each a dedicated port. Having multiple Web Listeners is a good way to balance the load among various applications.

The language that the Oracle Web Listener uses to communicate with its clients is HyperText Transfer Protocol (HTTP). The Web Listener implements version 1.0 of the HTTP protocol as described in the IETF HTTP Working Group (http:/ / www.ics.uci.edu/ pub/ ietf/ http/). All

MAC003085

ORCL00941761

Web clients must be able to speak the HTTP protocol in order to send and receive hypermedia documents.

HTTP is an application level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. It is a generic, stateless, object–oriented protocol which can be used for many systems, through extension of its request methods (commands).

Oracle Web Listener Features

The Oracle Web Listener includes a number of features to enhance performance and extend its usability in a variety of environments.

High Performance Architecture

For maximum performance the Web Listener is designed to run as an asynchronous engine via a single process with a single thread, in comparison to most HTTP servers, which start a new thread or process whenever a new connection is made. This considerably decreases the time required to service a request and reduces Listener machine resource utilization, providing high performance under a heavy load.

File Caching in Memory

The Oracle Web Listener allows a configurable set of commonly accessed files to be cached in memory. This provides very good performance when these files are accessed by a client.

In the current version of the Oracle Web Listener, the list of files to be cached in memory is determined by the WebServer Administrator and configured manually. The administrator should periodically analyze the Listener logs to determine which files should be cached and to change the configuration accordingly.

Memory Mapping of Files

On operating systems which support this feature, the Oracle Web Listener will automatically map files into memory addresses when they are accessed (unless it is otherwise specified that they are to be cached permanently into memory),. In this case the file will appear to be loaded in memory, allowing more than one connection accessing the same file to avoid duplicate disk reads. In addition, the operating system will

MAC003086

ORCL00941762

usually pre–fetch the next segment of a memory–mapped file while the previous one is being transmitted, further increasing performance, even in the single connection case.

Directory Mapping

The Oracle Web Listener supports mapping pathnames that appear externally in URLs on the Web to locations internally on the Web Listener machine. This allows a consistent appearance to the clients, even if the Web Listener's disk is reorganized.

The specific mapping of externally visible (virtual) directories to local physical directories in the file system is configurable by the WebServer Administrator.

For example, the URL

```
http://www.acme.com/products/info.html
```

can be mapped into the directory /disk1/products of system www.oracle.comas:

```
/disk1/products/info.html
```

If the products directory is moved to /disk3 in a reorganization, the Administrator reconfigures the virtual directory products to point to the new location /disk3/products in the Web Listener configuration file (owl.cfg), and the Web client is completely unaware of the change.

For a full description of the use of directory mappings to control the use of "virtual" parent directories and subdirectories, see the section "Directory Mappings" in Chapter 7, "The Oracle WebServer Administration Utility".

> Note: The Web Listener needs to be reloaded to recognize the change. For information on reloading the Web Listener, see "Web Listener Control Utility" later in this chapter.

Language Extensions

The Oracle Web Listener supports files stored in different character sets appropriate to the language being used. The Web Listener can interpret the file extension to provide both the type of the data in the file (such as HTML data) and the national language in which the file is written. The language mapping feature is controlled by the Administrator, and allows files written in different character sets appropriate to national languages to be accessed by clients of the Web Listener.

MAC003087

ORCL00941763

The mapping of specific file extensions to language and data types is controlled by the Web Listener configuration file.

For more information about the Web Listener configuration file, see "Configuration Parameters," later in this chapter.

HTTP Protocol Negotiation Features

One of the features of HTTP is the ability of clients and HTTP servers to exchange information about the types of data they can transfer and *interpret*. *These negotiation features allow the construction of systems* composed of different data types and languages.

When a Web client requests an object, it can specify what types of data it can receive and express a preference for specific types. The Web Listener can use the information to send the appropriately formatted data to the client if more than one format is stored on the Web Listener. This process is called HTTP type negotiation and is fully supported by the Oracle Web Listener. For example, a document might contain an inline image, which could be transferred in either GIF or JPEG format to a Web client.

The Oracle Web Listener also supports a similar concept for negotiation of the appropriate national language to display a document for a specific user. The document could be stored in multiple national languages (for *instance English and Canadian French), and the appropriate copy of the* document would be sent if the client expressed a preference. This makes the maintenance of a multilingual Web site a great deal easier, since there is no need to maintain duplicate URLs and separate directory trees for different languages.

Common Gateway Interface (CGI)

Common Gateway Interface (CGI) version 1.1 is the standard technique used by an HTTP server to execute a program that generates HTML output. This technique provides dynamic content rather than static content from files on disk. The Oracle Web Listener is fully compliant with CGI 1.1.

When a request comes in that the Web Listener recognizes as a request to execute a CGI application, a separate process is created to perform the operation. The Web Listener maintains communication with the process to transmit input from the Web client to it, and to retrieve the HTML output it may generate for display on the client's screen.

The Web Listener Administrator can configure specific directories to contain CGI applications to be run instead of files to be transmitted. Either of these can be done using the Web Listener configuration file. For

MAC003088

ORCL00941764

more information on the Web Listener configuration file, see "Configuration Parameters," later in this chapter.

CGI Application Execution

When a client requests a URL from the Oracle Web Listener, the URL's pathname component is analyzed to determine if it represents a directory containing files (the usual case). These files are then sent immediately to the client or to a directory containing CGI applications, sometimes referred to as scripts, which must be executed as separate programs to generate output which is then sent to the client.

The technique of CGI application execution is one of the key features of the Web. It allows the integration of many information sources. The Oracle Web Agent is one CGI application that can be accessed using this technique.

Once the Listener has determined that a URL represents a CGI application, it interprets the URL to extract path information and arguments to be passed to the CGI application on startup.

URLs to CGI programs are split into three different parts:

- virtual path
- extra path information
- query string

The syntax is as follows:

virtual_path extra_path_information?query_string

virtual path	similar to a path you would use to access a regular document or image. That is, it points the server to the file that contains the CGI program you want executed.
extra path information	additional information you can embed in the URL after the program name. Extra path information is optional. It can be used to convey constant information to your scripts independent of the client's intervention. It can also be used to access the server's virtual–to–physical path translation mechanism.
query string	another optional part of the URL. It can either be explicitly given in the hypertext anchor, it can come from a user typing into a search dialog box for an HTML document with the ISINDEX tag, or it can come from HTML forms.

MAC003089

ORCL00941765

User input from URLs and other resources

When data is typed into a search dialog or text entry form, the data is encoded using URL encoding. In this encoding the following rules apply:

- Spaces are changed into + signs
- Any of the characters can be "escaped" by changing them into a sequence of the form %xx, where x is a hexadecimal digit. The character is identified by translating the two hexadecimal digits into a number from 0–255, which represents a character.

If the data is coming from a search dialog resulting from an ISINDEX tag, the above translations can be applied directly. Further, your CGI program can receive this information fully translated on the command line if you want to avoid performing the translation yourself.

If the data is coming from an HTML form, then the location of this data varies depending on the method attribute specified with the FORM tag in your HTML document. If the GET method is used, this information comes from a QUERY_STRING variable. If the POST method is used, this information is sent to your program using standard input.

The data provided will be in the form:

name1=value1&name2=value2. . . . . &nameN=valueN

If there are any equal signs (=) or ampersands (&) in the encoded data, they are encoded using the above encoding rules. To decode the data appropriately, the NAME=VALUE pairs should be split (eliminating the ampersands), then each pair should be split into a name and a value, and then the URL decoding should be applied to each portion of the pair.

When a form is submitted you can often use the order in which the form items appear to determine the order in which your CGI program receives the NAME=VALUE pairs. However, you should not depend on this behavior. The various form elements have their own ways to determine what value will be associated with the name they are given. All of the textual input areas use the user's typed input as the value.

- Radio buttons use the value of the enabled button.
- Checkboxes are unchecked. Hence, they will use either an empty string, or their name will not appear in the encoded form data at all.
- Hidden form elements can be used to send constant or state information to your script.

MAC003090

ORCL00941766

Imagemap Support

The imagemap feature of the Oracle Web Listener allows a displayed image sent to the client to have multiple link areas that provide different hypermedia links (URLs) based on where the user clicks on the image. An example is a map of the world that allows the user to select a country by clicking on it. Use of imagemaps is extremely common in sophisticated Web sites.

The Oracle Web Listener includes native processing of imagemaps within the Web Listener. Many other HTTP servers require a separate process to be created to interpret an imagemap. By doing this internally, the Oracle Web Listener improves performance and lowers system resource utilization when this common feature of HTML is being used.

Domain Name Server (DNS) Resolution

When a client requests a URL from an HTTP server, the client's network address (IP address) is logged by the HTTP server and can be used to control access to documents on the HTTP server. The Domain Name System (DNS) converts the client's network address to the client's hostname as an ASCII string.

> Note: The DNS process can be time consuming, since another machine where the naming system is already running may be contacted.

The Oracle Web Listener provides three options for controlling DNS resolution of network addresses:

- Always resolve addresses on connections.
- Do "lazy" resolutions where the name is not resolved until an external application, a security module, or a CGI script needs it.
- Never resolve addresses even if a client asks for it.

You can use the Oracle WebServer Administration Utility or edit the configuration file to control which type of DNS resolution is used. The default is never to resolve using DNS, which provides the best performance.

When DNS resolution does occur, the results are cached. Since there are often several connections from the same client in close succession, this helps to minimize the performance penalty if DNS resolution is required.

MAC003091

ORCL00941767

Client to Web Listener Security

The security framework for the Oracle Web Listener includes support for Basic Authentication and Digest Authentication. Both of these schemes provide for username and password authentication on documents. Access may also be controlled based on the client's network address. Use the Oracle WebServer Administration Utility, described in Chapter 7, to configure the security features.

Basic Authentication

This scheme uses usernames and passwords which are sent in the clear across the network. This technique has largely been replaced in practice by Digest Authentication and is included for compatibility with older Web browsers that don't implement Digest authentication.

Digest Authentication

This scheme is similar to Basic Authentication, except that it uses usernames and passwords which are sent encrypted across the network. A cryptographic checksum (digest) is used to encode the password, preventing an attacker from gaining access to the original username/ password combination. This technique is significantly more secure than Basic Authentication and is recommended for all new applications, although older browsers, some of which are still quite popular, do not support Digest Authentication.

Internet Protocol–Based Restriction

The network address of the client may be used to restrict access to information on the Oracle Web Listener. Addresses may be individually restricted or allowed access, or entire IP networks may be specified to allow users within certain locations access while denying all others.

When a client requests a file protected by IP–based security, its address is compared against the specified addresses in the order given, and as soon as a match is found, the client is allowed or denied access as specified. Files protected by IP–based restriction may not be accessed by clients other than those listed in the configuration parameters.

Note that Digest Authentication is a safer technique to determine who a user really is, since a credential (the password) must be presented to gain access. Network address–based restriction techniques are very convenient in that they avoid password management problems, but are also less secure, since a clever attacker may falsify network addresses.

Domain Name–Based Restriction

Domain name–based restriction is similar to IP–based restriction. A list of groups and hostnames specifies which hosts should or should not have access to a set of documents.

This technique uses symbolic hostnames rather than network addresses. Hence, it is easier to administer since network addresses may change if the network architecture is changed. This technique still suffers from the

MAC003092

ORCL00941768

problem of clever attackers falsifying their network addresses and assuming the identity of a host other than their own.

Security on Specific Files Specific files can be made secure by associating them with Basic Authentication, Digest Authentication, IP–based restriction, domain–name based restriction, or any combination of one authentication and one restriction method.

Other Security Features In addition to the authentication features, the Web Listener's user and group ID (on UNIX implementations) may be specified in the configuration file so that the Web Listener itself can be restricted to accessing only files available to that user and group. This allows the Web Listener to be started as a privileged user as required on some implementations, but still operates as if it were an unprivileged program. This provides additional security against an attack directly on the Web Listener process. For more information on the use of the Group ID for security purposes, see "Basic Configuration Parameters" in Chapter 7 of this document.

Logging

When a request is made of the Web Listener, the request information may be logged to a specified log file. The Oracle Web Listener records connections in the industry standard Common Log Format. This format includes the first line of the client's request, which specifies the action, URL, and protocol version. It also includes such information as the client's host address, whether the request was successful, and how many bytes were transmitted. This information may be later used to analyze what types of accesses are performed by which users, and to check for errors. Use the WebServer Administration Utility to specify the log file directory and the names of the standard and error log files in the Web Listener configuration file.

> Note: Be sure that the user ID under which the Web Listener runs has privileges to read and write to the log file directory.

Starting Up and Shutting Down the Oracle Web Listener

The easiest way to start up, shut down, or reload an Oracle Web Listener is to use the Administration Utility described in Chapter 7 of this document. If you wish to perform these operations from the UNIX operating system, use the WLCTL Utility as described in the next section.

MAC003093

ORCL00941769

The WLCTL (Web Listener Control) Utility

This utility resides in the ORACLE_HOME bin directory and allows you to easily startup, stop, and reload your Oracle Web Listener.

The syntax you use to invoke WLCTL to perform various operations is as follows:

```
> wlctl [start|stop|reload] [port_number] ORACLE_HOME
```

If ORACLE_HOME is set in your environment and is the ORACLE_HOME where the Oracle WebServer is installed, it need not be included on the command line.

The following examples assume that the Oracle WebServer is located in /u01/oracle:

- With ORACLE_HOME set:

  ```
  >wlctl start 8888
  ```

- Specifying ORACLE_HOME on the command line:

  ```
  >wlctl start 8888 /u01/oracle
  ```

- With ORACLE_HOME set:

  ```
  wlctl stop 8888
  ```

- Specifying ORACLE_HOME on the command line:

  ```
  wlctl stop 8888 /u01/oracle
  ```

- With ORACLE_HOME set:

  ```
  >wlctl reload 8888
  ```

- Specifying ORACLE_HOME on the command line:

  ```
  >wlctl reload 8888 /u01/oracle
  ```

The stop and reload functions require that the Listener PID file for the Web Listener you are stopping or reloading resides in the location pointed to by the Web Listener configuration file.

Oracle Web Listener Configuration Parameters

The configuration parameters for the Oracle Web Listener are stored in owa.cfg, the Web Listener configuration file. It is read when the Web Listener is initially started and on receipt of a signal '1' on UNIX implementations. The Web Listener configuration file is specified with the –c option on the command line when the Web Listener is started,

MAC003094

ORCL00941770

allowing multiple Web Listeners with different configuration files to be started on the same Web Listener machine.

Oracle WebServer provides an HTML–based Administration Utility which may be accessed with any forms–capable Web browser. This utility eliminates the need for the WebServer administrator to edit the Web Listener configuration file manually in most cases, and includes explanatory help text on the individual parameters. The Administration Utility is described in Chapter 7, "The Oracle WebServer Administration Utility". This section documents the parameters in the configuration file for completeness, in the event an administrator wishes to edit the file manually.

The Web Listener configuration file is divided into sections, which start with a section name in brackets—for example, [NetInfo]. Individual configuration parameters are set by name = value pairs, with the configuration parameter on the left of the equal sign and the value on the right as specified in the table below.

For example, consider the following portion of a configuration file:

```
;
;  www.acme.com configuration file
;
[NetInfo]
HostName  =  www.acme.com
HostAddress  =  ANY
PortNumber  =  80
```

This portion of the file sets HostName to www.acme.com, HostAddress to ANY and PortNumber to 80.

Table 3 – 1 contains the parameters, grouped by logical function.

Section	Parameter Name	Default Value	Description
NetInfo	HostName	none	hostname Web Listener is on, using the fully qualified host name, e.g.. www.acme.com The Listener uses the primary Internet address if no hostname is specified.
	HostAddress	ANY	Internet address on which the Web Listener on a multi–homed host accepts connection. ANY indicates connections accepted on all interfaces (IP addresses). Leave this set to ANY unless you have a specific need to change it.

MAC003095

ORCL00941771

PortNumber	80	TCP/ IP port number on which the Web Listener accepts requests. Note that port numbers less than 1024 require the Web Listener to run as root on many UNIX systems. 80 is the usual default for Web HTTP servers.
MaxConnectCount	50	Maximum number of simultaneous connections from users. The Web Listener ignores requests over this limit. Note there is a compile–time per–platform limit which may not be exceeded, no matter what value is specified here.
DNSResolution	NEVER	Controls DNS address to name resolution. If set to ALWAYS, the Web Listener always translates address to names. If set to LAZY, the Web Listener resolves names on demand only. If set to LAZY_WITH_CGI, the Web Listener also resolves on demand for CGI applications. If set to NEVER the Web Listener never resolves, even on demand (this provides the best performance).
SERVERPID	none	Name of file in which to store the Web Listener's process identifier (PID).

Table 3 – 1 Oracle Web Listener Network Configuration Parameters

Logging Configuration Parameters

Table 3 – 2 contains the Oracle Web Listener logging parameters.

Section	Parameter Name	Default Value	Description
[Log]	LogDir	none	Name of directory to contain log files. This directory must exist before the Web Listener starts.
	LogInfoFile	none	Name of file in which to store routine log information.
	LogErrorFile	none	Name of file in which to log errors.

Table 3 – 2 Oracle Web Listener Logging Configuration Parameters

MAC003096

ORCL00941772

Note: Be sure that the user ID under which the Web Listener runs has privileges to read and write to the log file directory.

Directory Mapping Parameters

Directories are mapped individually in the [DirMaps] section. Each line looks like the following:

Directory_Name R|N|C *Virtual_Path_Name*

- The directory name on the left is the physical directory name on the Web Listener file system.
- The choice of R|N|C controls how the directory name is interpreted.
  - R means to map 'recursively', which means the directory is a tree of subdirectories, all of which are mapped into a tree starting at the specified virtual path name.
  - N means to map only the single directory specified, and no subdirectories.
  - C means that the directory contains CGI applications instead of files to be transmitted.
- Virtual_Path_Name is the location this directory should appear on the Web.

Consider the follow example of the directory mapping section of a Web Listener configuration file:

```
;
; directory mapping section of Web Listener configuration file
;
[DirMaps]
c:\html    R      /
```

For instance, in the above example, the directory c:\html and all subdirectories below it are mapped to the root directory (/) on the Web. So an access to the URL:

http://*listenername*/index.html

would actually read the following file on the Web Listener's disk.

c:\html\index.html

File Cache Definitions

This section of the configuration file, starting with the key "[FileCache]" is simply a list of all the files that should be cached in memory. This

MAC003097

ORCL00941773

causes the Web Listener to keep the file open for the life of the Web Listener.

If a file is not listed here, it is not cached. When a non–cached file is requested by a client, the server maintains the resources to access that file only as long as the client refers to the file. The server will release the resources required to access a file requested by a client once there are no outstanding client references to the file.

There are three options for specifying files in the file cache.

- An explicit filename can be given.
- An incomplete filename ending in an asterisk can be used to indicate that all files that match the portion before the asterisk will be cached.
- A directory can be specified, in which case all its files and subdirectories will be cached.

The following example specifies that the specific file `index.html`, all files starting with `a` in the directory `/marketing`, and all files in the directory `/products` and its subdirectories should be cached.

```
;
; File Cache Definitions
;
[FileCache]
/index.html
/marketing/a*
/products
```

> Note: The filenames specified are virtual pathnames (the names that appear in URLs on the Web), not necessarily the physical names of the files on disk if the Directory Mapping feature has been used.

Encoding Definitions

This section contains a list of file encodings that the Web Listener understands, and the file extensions that indicate a file is of the given type.

For example:

```
;
; encoding definitions
; List for each combination of encodings
; one or more unique extensions
;
[Encodings]
```


MAC003098

ORCL00941774

```
compress        Z
gzip            gz
```

MIME Types Definitions

This section of the configuration file defines what Multimedia Internet Mail Extensions (MIME) file types the Web Listener recognizes, and how it maps the extensions of specific files to those types. This allows the proper information about file type to be transmitted to the client with the file data, which results in proper interpretation of the file on the client. The client may spawn a helper application to interpret the file if the type is not interpreted natively by the browser in use. This process is controlled entirely on the client side.

Note that files that do not map to any type are by default considered MIME type application/octet–stream which denotes a binary file.

Each line in this section consists of:

MIME–type list of file extensions mapping to that type

For example:

```
[MIMEType]
text/html       htm html
image/jpeg      jpg jpeg
image/gif       gif
```

This example tells the Web Listener that files with extensions:

- htm and html are treated as MIME type text/html
- jpg and jpeg indicate an image/jpeg file
- gif indicates it is an image/gif file

These are all common MIME types that clients understand. There are many specialized types as well for conveying other forms of data, such as audio or video clips, and so forth.

Language Extensions

This section defines filename extensions that the Web Listener recognizes and the languages and character sets to which those extensions are mapped. Extensions are case sensitive. Note that each mapping conveys both a language and a character set mapping. All language extensions that differ only in character set are equivalent for non–text files. The ability to specify a language and a character set is

MAC003099

ORCL00941775

important for languages which have more than one character set representation (for example, Arabic).

Each line consists of:

language_ID character_set_ID list_of_file-extensions

For example:

```
[LangExt]
en iso-8859-1   eng
en unicode-1-1  engUuc
fr-CA  iso-8859-1      frc
```

In this example, all files with the extension `eng` are mapped to English, ISO 8859–1 character set. Files with extension `engU` or `uc` are mapped to English, in the unicode character set. Files with extension `frc` are mapped to French Canadian language, ISO–8859–1 character sets.

The names of the character sets are as specified in RFC 1521, the language identifiers are defined in RFC 1766.

In order to use language mapping capability within the Web Listener, URLs must be specified without trailing extensions. The Web Listener adds the appropriate language/ character set extension as required if the client supports natural language negotiation.

For instance:

`http://www.oracle.com/index`

results in access to the file:

`/index.frc.html`

if the client had configured French Canadian as the preferred language.

A list of languages may be configured on the client, so that several languages may be specified and if any of them is present, the file will be sent according to the first match. Note that if a file has no language extension before its type extension, that file is assumed to be language neutral. The default character set is ISO–8859–1.

To summarize: if the files `/index.frc.html` and `/index.html` existed, users who specified French Canadian as a language preference would see the first version of the file, and all other users would see the second version.

MAC003100

ORCL00941776

Basic and Digest Authentication Configuration

This section of the configuration file specifies the type of authentication accepted for a certain file access, and the parameters for that authentication type, including usernames, passwords, and file names.

There are several sections in the file, one for each type of security.

The concept in Basic and Digest authentication is the same. Users are listed in username/ password combinations, followed by specifying which users are in what groups. Finally, groups are assigned to security realms which are the parameters assigned to specific files to protect them.

Basic authentication is configured identically to Digest authentication, with the module name `Digest` above replaced with `Basic`.

Digest authentication is recommended wherever possible, to eliminate the possibility of passwords being intercepted on the network.

An example:

```
[Security]
        Digest {
                (Users)
                jjones: glorkz
                kmaco:  plugh
                ajames: foo

                (Groups)
                mktg:   ajames
                dev:    kmaco
                all:    jjones kmaco ajames

                (Realms)
                restricted:     all
                secret:         dev mktg
        }
```

This segment defines three users, (`jjones`, `kmaco`, `ajames`), with passwords (`glorkz`, `plugh`, `foo`) respectively. The group `all` has everyone in it, the groups `mktg` and `dev` have a single individual in each. The security realm `restricted` is accessible to all three people, but only the members of groups `dev` and `mktg` can access the `secret` realm.

In this situation, any file that was defined in the `restricted` realm would be accessible to the three users specified here (but no one else who used the same Web Listener), and any file in the `secret` realm would be accessible only to users `ajames` and `kmaco`.

MAC003101

ORCL00941777

Network Address (IP) and Hostname Restriction Configuration

IP and hostname restriction is configured more simply. Hosts or IP addresses are assembled into groups, which can have specific files assigned to their access privileges.

To include a host or a group of hosts, specify the plus sign (+) before the identifier for that host and exclude it by including a minus sign (–) before the identifier. In addition, the asterisk wildcard (*) can be used to match groups of hosts.

For example:

```
[Security]
        IP {
                st:     +144.25.16.*
                        +144.25.20.*
                dc:     +130.35.*.*
                        –130.35.1.1
        }
```

This would configure everything on the networks 144.25.16 and 144.25.20 to be in group st , and everything on network 130.35 except host 130.35.1.1 to be in group dc. If the client's address doesn't match any of these, access will be denied to any file specified to require st or dc security access.

The same technique is used with hostnames in the case of domain name restriction.

The following example configures all machines in the domain oracle.com to be in group oracle except machine www.oracle.com

```
[Security]
        Domain {
                oracle: +*.oracle.com
                        –www.oracle.com
        }
```

Once the security groups have been defined, you can assign specific protections to individual files.

Assigning Security to Individual Files

When you assign protections, you specify the filename—possibly with wildcard characters to specify a group of files, or a directory name to include that directory and everything under it—followed by a list of security schemes and a parentheses-enclosed string. The string is a realm name for Digest or Basic authentication, and a group name for IP or domain restriction. The types may be combined, by requiring either scheme to be satisfied if the vertical bar (|) is used to separate security

MAC003102

ORCL00941778

schemes, or BOTH schemes must be satisfied if the ampersand (&) is used. If a combination of security schemes is used, one scheme must be an authentication type, and the other a restriction type.

For example:

```
[Protection]
/secret/*          IP(st) | Basic(secret)
/company/*         Domain(acme)
```

This example protects everything under the directory /secret using either IP–based restriction from the st group or by anyone who can log into the secret realm using Basic authentication. All the files in /company are protected so that only people in the acme domain may see them.

Oracle Web Listener Miscellaneous Configuration Parameters

Section	Parameter Name	Default Value	Description
Server	UserDir	none	Directory under a user's home directory searched when the URL /~username/ is received. This allows users to store their own home pages in their home directories (UNIX only).
	InitialFile	initial	The file that the Listener looks for when a URL ending in a directory rather than a file is sent to the Listener. This is the 'index' file for the Web Listener.
	UserDirInitialFile	initial.html	Default filename when the /~username/ construction is used in a URL with no file specified.
	DefaultMIMEType	application/octet–stream	Default MIME type used by the Web Listener when the filename extension requested in a URL is not recognized.
	DefaultCharset	iso–8859–1	Default character set if none is inferred from the file extension.
	PreferredLanguage	en	Language a Web Listener will prefer given a choice of languages when searching for a file. Default is English (en).

ImageMap	none	Set this to 'map'. This is the extension the Web Listener expects imagemaps to have.
ServiceTimeout	none	On Windows NT only: Specifies the time in seconds that the NT control panel will wait for a pending start, stop, or continue command to complete before reporting a failure.

Table 3 – 3 Oracle Web Listener Miscellaneous Listener Configuration Parameters

MAC003104

ORCL00941780

CHAPTER

4

Introduction to the Oracle7 Server and to PL/ SQL

This chapter provides an overview of the Oracle7 Server. It covers the basics of relational databases, SQL, and PL/ SQL. If you already are familiar with some or all of this material, you can reasonably skip part or all of this chapter. Since there is more material on these subjects than a chapter can possibly cover, there are frequent references to other Oracle manuals that provide you with further information.

What is the Oracle7 Server?

The Oracle7 Server is a Relational Database Management System (RDBMS). That is to say, the job of the Oracle7 Server is to manage data. Users or other processes store, alter, and obtain the data by issuing statements that the RDBMS executes, but they never directly access the data. Having all the data under the control of a single entity enables that entity to ensure, for example, that the data maintains a coherent structure and that simultaneous changes by different users do not interfere with one another.

MAC003105

ORCL00941781

Database Tables

Saying that Oracle7 is a Relational Database Management System implies that all of the data it contains is structured as tables (tables are called "relations" in mathematical jargon). Even the metadata—the internally generated information describing the content of the database, such as how many tables there are and what they are called—is contained in a special set of tables called the Data Dictionary (for more information on the Data Dictionary, see Chapter 1 of the Oracle7 Server Concepts Manual).

Here is a simple table, such as you might find in an Oracle database:

CNUM	FNAME	LNAME	ADDRESS
4005	Julia	Peel	197 Myrtle Court, Brisbane, CA
4007	Terry	Subchak	2121 Oriole Way, Boston, MA
4008	Emilio	Lopez	31D San Bruno Ave. SF, CA
4011	Kerry	Lim	455 32nd St. #45, Brinton, KY

Table 4 – 1 Customers

Each row of this table describes one person, and each column has one type of information about that person. Note the column cnum. This is simply a number we generate to distinguish the customers from one another, as names are not necessarily unique. As you will see shortly when we discuss SQL, you refer to data in a relational database by its content, not by such things as where it is stored. Therefore, every table must have an identifying group of one or more columns whose values, taken as a set, are always different for every row of the table. This group, in this case the single column cnum, is called the primary key of the table. Locally generated numbers, as in this example, are a common and easy way to create primary keys.

Foreign Keys

Suppose we wanted to add our customers' phone numbers to our database. Since one person can have multiple phone numbers, these do not fit into our structure well. If we want to include the phone numbers in the Customers table, there are three possibilities, none of them good:

MAC003106

ORCL00941782

1. You could fit all the phone numbers for a given person into a single column in a single row, in which case it would be difficult to access the phone numbers independently.
2. You could create a column for each type of phone number, in which case you would have to redesign your table each time a new type arose. This also could create an unwieldy number of mostly empty columns.
3. You could enter a new row for each phone number, in which case each such row would consist of redundant information except for the new phone number. This approach would be error–prone and waste space.

The good solution is simply to create a second table, like this:

CNUM	PHONE	TYPE
4005	375–296–8226	home
4005	375–855–3778	beeper
4008	488–255–9011	home
4011	577–936–8554	home
4008	488–633–8591	work

Table 4 – 2 Customers_Phone

Notice that in this table cnum is not the primary key; it identifies the customer and therefore is the same for each phone number associated with a given customer. What, then, is the primary key? The combination of cnum and phone. If we list the same number for the same person twice, we really have made a duplicate entry and should eliminate one anyway.

The cnum column does have a special function, however, because it defines the relationship between Customers_Phone and Customers by associating each phone number with a customer. We say that it references the cnum column in Customers. A group of one or more columns, such as this, that references another group is known as a foreign key. The group of columns a foreign key references is called its parent key or its referenced key. Each foreign key value references a specific row in the table containing the parent key. Clearly, then, all sets of values in the foreign key have to be present once and only once in the parent key (although they may be present any number of times in the foreign key itself, as above) for the reference to be both meaningful and unambiguous. For that reason, the parent key must be either a primary key (the usual case) or another group of columns that is unique, which is known as a unique key.

MAC003107

ORCL00941783

Oracle can make sure that all primary and unique keys stay unique and that all foreign key references are valid; this is called maintaining referential integrity. For more information on foreign and parent keys, see Chapter 7 of the Oracle7 Server Concepts Manual and "CONSTRAINT clause" in Chapter 4 of the Oracle7 SQL Reference.

Although Oracle tables may look similar to, and often be presented as, HTML tables, there is a subtle but important difference. In HTML, tables are a way of presenting data to the user. The data itself may not actually have a tabular structure or any structure at all. For example, since not all browsers can interpret graphics, you could represent a bar graph in HTML as a table.

In the Oracle RDBMS, tables are how all data is structured and stored. Once the data from the tables is retrieved by other processes, however, those processes can reformat it so that it is not presented to the users as tables at all. For example, a very simple two–column table might consist of graphics (Oracle can store graphics, sound, video, or any binary value in a table using the datatype LONG RAW) in one column and a name for each graphic in the other. An application could use the names to request the graphics it wants from the database and present the graphics to the user, never revealing that they came from a table.

Users, Connections, Privileges, and Roles

The Oracle7 Server controls which users can do what to its data. To do this, it uses a log on procedure that is separate from that of the operating system. Once logged on the operating system, you establish a connection to the Oracle7 Server with a username and password known to Oracle that may have no relationship to the ones used for you by the operating system. You can also be Identified Externally, in which case you still connect to the database with an independent username, but Oracle7 uses the operating system to verify your identity, rather than requesting another password. For more information, see "CREATE USER" and "CONNECT" in the Oracle7 Server SQL Reference.

The name under which you connect to Oracle7—your Oracle username— is associated with a number of privileges, which are the rights to perform various actions. Privileges can be granted and revoked dynamically, and sometimes you can grant privileges you have received to other users as well. Some privileges apply to specific database objects—these are called object privileges. Others are more general in nature—these are called system privileges. For more information on privileges, see "GRANT" and "REVOKE" in Chapter 4 of the Oracle7 Server SQL Reference.

MAC003108

ORCL00941784

Roles are groups of privileges that are granted and revoked as single units. They simplify complex privilege assignments. They also differ from simple privileges in that they are not always in force. You can be granted a role, but have it disabled, in which case you cannot use the privileges the role contains until you enable it. For more information on roles, see "CREATE ROLE", "ALTER ROLE", and "GRANT" in the Oracle7 Server SQL Reference.

User actions can also be constrained by profiles. These control such things as how many simultaneous connections you can have and how long they can last. Profiles are beyond the scope of this manual. For more information on profiles, see "CREATE PROFILE" in the Oracle7 Server SQL Reference.

Distributed Databases

If you have a full license for the Oracle7 Server, you can use it to create distributed databases. These are multiple Oracle7 databases—located on different machines and possibly separated by considerable geographical distance—that work together to maintain a single consistent body of data that looks and behaves as though drawn from a single source. In this situation, you connect to one database, which is called the local database, and that database interacts with other databases, called remote databases, as needed. The mechanism used to refer to objects on a remote database is called a database link or dblink. For more information on distributed databases, see "Distributed Databases" in the Oracle7 Server Concepts Manual.

The software that enables distributed databases to communicate with one another is SQL*Net, an Oracle product that enables communication between various servers that possibly are on different platforms and using different network protocols. For more information on SQL*Net, see Understanding SQL*Net.

What is SQL?

SQL (Structured Query Language) is the language you use to issue instructions to the Oracle7 Server. It is, in fact, the standard language used by all major relational database vendors, and Oracle complies at Entry Level with SQL92, the most recent ISO (International Standards Organization) standard. There are several aspects of SQL that may differ from computer languages that you are familiar with, such as the following:

- SQL is non–procedural. In SQL, you tell the Server what to do but not how it is to be done. This frees you from dealing with a lot of detail.
- SQL statements are independent of one another. Although PL/ SQL (discussed later in this chapter) addresses this, SQL itself has no conditional or other control–flow statements.
- SQL employs set–at–a–time operation. It operates on arbitrarily large sets of data in a single step.
- SQL uses three–valued logic. In most languages, Boolean expressions are either TRUE or FALSE. In SQL, they are TRUE, FALSE, or NULL. This will be explained shortly.

Retrieving Data

Suppose you wanted to pull from the Customers table the information on customers named "Peel". This is called making a query. To do it, you could issue the following statement:

```
SELECT *
    FROM Customers
    WHERE LNAME = 'Peel';
```

This produces the following:

```
CNUM  FNAME       LNAME   ADDRESS
4005  Julia       Peel    197 Myrtle Court, Brisbane, CA
```

Oracle interprets the statement as follows: Any number of spaces and/ or line breaks are equivalent to one space or line break. These are delimiters, and the extra spaces and line breaks are for readability: all are equivalent "white space". Likewise, case is not significant, except in literals like the string you are searching for ('Peel').

SELECT is a keyword telling the database that this is a query. All SQL statements begin with keywords. The asterisk means to retrieve all columns; alternatively, you could have listed the desired columns by name, separated by commas. The FROM Customers clause identifies the table from which you want to draw the data.

WHERE LNAME = 'Peel' is a predicate. When a SQL statement contains a predicate, Oracle tests the predicate against each row of the table and performs the action (in this case, SELECT) on all rows that make the predicate TRUE. This is an example of set–at–a–time operation. The predicate is optional, but in its absence the operation is performed on the entire table, so that, in this case, the entire table would have been retrieved. The semi–colon is the statement terminator.

MAC003110

ORCL00941786

Nulls and Three–Valued Logic

With predicates, you should be aware of three–valued logic. In SQL, the basic Boolean values of TRUE and FALSE are supplemented with another: NULL, also called UNKNOWN. This is because SQL acknowledges that data can be incomplete or inapplicable and that the truth value of a predicate may therefore not be knowable. Specifically, a column can contain a null, which means that there is no known applicable value. A comparison between two values using relational operators—for example, a = 5—normally is either TRUE or FALSE. Whenever nulls are compared to other values, however, including other nulls, the Boolean value is neither TRUE nor FALSE but itself NULL.

In most respects, NULL has the same effect as FALSE. The major exception is that, while NOT FALSE = TRUE, NOT NULL = NULL. In other words, if you know that an expression is FALSE, and you negate (take the opposite of) it, then you know that it is TRUE. If you do not know whether it is TRUE or FALSE, and you negate it, you still do not know. In certain cases, three–valued logic can create problems with your programming logic if you have not accounted for it. You can treat nulls specially in SQL with the ISNULL predicate, as explained in Chapter 3 of the Oracle7 Server SQL Reference.

Creating Tables

This is how you create tables in SQL. You can use the following SQL statement to create the Customers table:

```
CREATE TABLE Customers
(cnum       integer  NOT NULL PRIMARY KEY,
FNAME       char(15)  NOT NULL,
LNAME       char(15)  NOT NULL,
ADDRESS             varchar2 );
```

After the keywords CREATE TABLE come the table's name and a parenthesized list of its columns with a definition of each. Integer, char, and varchar2 are datatypes: all of the data in a given column is always of the same type (char means a fixed and varchar2 a varying length string). For more information on SQL datatypes, see Chapter 2 of the Oracle7 Server SQL Reference.

NOT NULL and PRIMARY KEY are constraints on the columns they follow. They restrict the values you can enter in those columns. Specifically, NOT NULL *forbids you from entering nulls in the column.* PRIMARY KEY prevents you from entering duplicate values into the column and makes the column eligible to be the parent for some

foreign key. For more information, see "CREATE TABLE" and "CONSTRAINT clause" in Chapter 4 of the Oracle7 Server SQL Reference.

Ownership and Naming Conventions

Note that when you create a table in SQL, you own it. This means you generally have control over who has access to it, and that it is part of a schema that bears your Oracle username. A schema is a named collection of database objects under the control of a single Oracle user. Schemas inherit the names of their owners. When other users refer to an object you have created, they have to precede its name by the schema name followed by a dot (no spaces). SQL utilizes a hierarchical naming convention with the levels of the hierarchy separated by dots. In fact, you sometimes have to precede column names by table names to avoid ambiguity, in which case you also use a dot. The following is an example in the form schemaname.tablename.columnname:

```
scott.Customers.LNAME
```

You can simplify references like this by using synonyms, which are aliases for tables or other database objects. Synonyms can be private, meaning that they are part of your schema and you control their usage, or public, meaning that all users can access them. For example, you can create a synonym "Cust" for scott.Customers as follows:

```
CREATE SYNONYM Cust FOR scott.Customers;
```

This would be a private synonym, which is the default. Now you could rewrite the example above like this:

```
Cust.LNAME
```

You still have to refer to the column directly. Synonyms can only be for tables, not table components like columns.

For more information on synonyms, see "CREATE SYNONYM" in Chapter 4 of the Oracle7 Server SQL Reference. For more information on SQL naming conventions, see Chapter 2 of the Oracle7 Server SQL Reference. For more on schemas, see "CREATE SCHEMA" in Chapter 4 of the Oracle7 Server SQL Reference.

Inserting and Manipulating the Data

Which SQL statements determine the actual data content? Chiefly, three—INSERT, UPDATE, and DELETE. INSERT places rows in a table, UPDATE changes the values they contain, and DELETE removes them.

MAC003112

ORCL00941788

The INSERT Statement

For INSERT, you simply identify the table and its columns and list the values, as follows:

```
INSERT INTO Customers (cnum, FNAME, LNAME)
    VALUES (2004, 'Harry', 'Brighton');
```

This statement inserts a row with a value for every column but ADDRESS. Since you did not, in your CREATE TABLE statement, place a NOT NULL constraint on the ADDRESS column, and since you did not give that column a value here, Oracle sets this column to null. If you are inserting a value into every column of the table, and you have the values ordered as the columns are in the table, you can omit the column list. You optionally can put a SELECT statement in place of the VALUES clause of the INSERT statement to retrieve data from elsewhere in the database and duplicate it here. For more information on the INSERT and the SELECT statements, see "INSERT" and "SELECT", respectively, in Chapter 4 of the Oracle7 Server SQL Reference.

The UPDATE Statement

UPDATE is similar to SELECT in that it takes a predicate and operates on all rows that make the predicate TRUE. For example:

```
UPDATE Customers
    SET ADDRESS = null
    WHERE LNAME = 'Subchak';
```

This sets to null all addresses for customers named 'Subchak'. The SET clause of an UPDATE command can refer to current column values. "Current" in this case means the values in the column before any changes were made by this statement. For more information on the UPDATE statement, see "UPDATE" in Chapter 4 of the Oracle7 Server SQL Reference.

The DELETE Statement

DELETE is quite similar to UPDATE. The following statement deletes all rows for customers named 'Subchak':

```
DELETE FROM Customers
    WHERE LNAME = 'Subchak';
```

You can only delete entire rows, not individual values. To do the latter, use UPDATE to set the values to null. Be careful with DELETE that you do not omit the predicate; this empties the table. For more information on DELETE, see "DELETE" in Chapter 4 of the Oracle7 Server SQL Reference.

Querying Multiple Tables Through Joins

Even though it only retrieves data, SELECT is the most complex statement in SQL. One reason for this is that you can use it to query any number of tables in one statement, correlating the data in various ways. One way to do this is with a join, which is a SELECT statement that correlates data from more than one table. A join finds every possible combination of rows, such that one row is taken from each table joined. This means that three tables of ten rows each can produce a thousand rows of output (10 * 10 * 10) when joined. Typically, you use the predicate to filter the output in terms of some relationship. The most common type of join, called a natural join, filters the output in terms of the foreign key/ parent key relationship explained earlier in this chapter. For example, to see the people in the Customers table coupled with their various phone numbers from the Customers_Phone table, you could enter the following:

```
SELECT a.CNUM, LNAME, FNAME, PHONE, TYPE
    FROM Customers a, Customer_Phone b
    WHERE a.CNUM = b.CNUM;
```

In the above, a and b are range variables, also called correlation variables. They are simply alternate names for the tables whose names they follow in the FROM clause, so that a = Customers and b = Customers_Phone. You can see that here you need the range variables to distinguish Customers.CNUM from Customers_Phone.CNUM in the SELECT and WHERE clauses. Even when not needed, range variables are often convenient.

Here is the output of the natural join:

CNUM	LNAME	FNAME	PHONE	TYPE
4005	Peel	Julia	375–296–8226	home
4005	Peel	Julia	375–855–3778	beeper
4008	Lopez	Emilio	488–255–9011	home
4008	Lopez	Emilio	488–633–8591	work
4011	Lim	Kerry	577–936–8554	home

This output represents every combination of rows from the two tables *where both rows have the same CNUM value.*

Outer Joins

Notice in the preceding example that people from the Customers table who did not have phones (namely, CNUM 4007) were not selected. If a row has no match in the other table, the predicate is never true for that row. Sometimes, you do not want this effect, and you can override it by using an outer join. An outer join is a join that includes all of the rows from one of the tables joined, regardless of whether there were matches

MAC003114

ORCL00941790

in the other table. Such a join inserts nulls in the output in whichever columns were taken from the table that failed to provide matches for the outer–joined table. Here is the same query done as an outer join:

```
SELECT a.CNUM, LNAME, FNAME, PHONE, TYPE
    FROM Customers a, Customer_Phone b
    WHERE a.CNUM = b.CNUM(+);
```

This is the output of the above:

CNUM	LNAME	FNAME	PHONE	TYPE
4005	Peel	Julia	375–296–8226	home
4005	Peel	Julia	375–855–3778	beeper
4007	Subchak	Terry	NULL	NULL
4008	Lopez	Emilio	488–255–9011	home
4008	Lopez	Emilio	488–633–8591	work
4011	Lim	Kerry	577–936–8554	home

Notice that the only difference in the query is the addition of (+) to the WHERE clause. This follows the table for which nulls are to be inserted. The output from the query, then, includes at least one row for each row of the table that did not have (+) appended in the predicate.

You can also use SELECT statements to produce values for processing within queries (these are called subqueries), and you can perform standard set operations (UNION, INTERSECTION) on SELECT statement output. For more information on the SELECT statement, subqueries, and joins, see "SELECT" in Chapter 4 of the Oracle7 Server SQL Reference.

Where to Look for More Information

Oracle7 SQL is a very complex subject, and we have been able only to scratch the surface of it here. To make it easier for you to find the specific information you need to perform the task at hand, we provide the following table, which identifies where in the Oracle7 Server documentation set you can find information on specific SQL topics. Unless otherwise noted, find the headings in Chapter 4 of the Oracle7 Server SQL Reference.

To Find Out About	Look Under
aggregate data (totals, counts, averages, and so on)	SQL Functions in Chapter 3 of the Oracle7 Server SQL Reference
changing user passwords	ALTER USER
connecting to the database	CONNECT
constraints	CONSTRAINT clause; CREATE TABLE; ENABLE clause

controlling user access to objects and user actions	GRANT; REVOKE; CREATE ROLE; SET ROLE; see also Chapters 17 and 18 in the Oracle7 Server Concepts Manual
creating databases	CREATE DATABASE
creating users	CREATE USER
functions that change simple values	SQL Functions in Chapter 3 of the Oracle7 Server SQL Reference.
linking databases at different locations	CREATE DATABASE LINK; see also "Distributed Databases" in the Oracle7 Server Concepts Manual.
making changes to the data permanent	COMMIT; SET TRANSACTION; SAVEPOINT
making SQL statements execute more quickly	CREATE INDEX; see also "Indexes" in the Oracle7 Server Concepts Manual.
monitoring database usage	AUDIT
reversing (undoing) changes to the data	ROLLBACK; SET TRANSACTION; SAVEPOINT

Table 4 – 3 Guide to Further Information on SQL

What is PL/SQL?

As you may have noticed from the preceding section, SQL statements are very concise and powerful, but do not do more as a group than they do individually. Generally speaking, SQL statements operate independently, having little effect on one another. This is of limited use for writing programs, where you must create a body of code that is going to vary its behavior according to the data and to user or other input. To develop applications with SQL, you generally have to either interface it to a standard programming language such as C, or extend it so that it becomes a useful programming language in itself. Oracle supports both approaches, but the latter approach has many advantages that are relevant to the Web, and is therefore the approach that the Oracle WebServer takes.

PL/ SQL, then, is an application–development language that is a superset of SQL, supplementing it with standard programming–language features that include the following:

MAC003116

ORCL00941792

- block (modular) structure
- flow–control statements and loops
- variables, constants, and types
- structured data
- customized error handling

Another feature of PL/ SQL is that it allows you to store compiled code directly in the database. This enables any number of applications or users to share the same functions and procedures. In fact, once a given block of code is loaded into memory, any number of users can use the same copy of it simultaneously (although behavior is as though each user had her own copy), which is useful for the Oracle WebServer. PL/ SQL also enables you to define triggers, which are subprograms that the database executes automatically in response to specified events.

Unlike SQL, PL/ SQL is not an industry standard, but is an exclusive product of Oracle Corporation.

> Note: For the sake of efficiency, PL/ SQL code is compiled prior to runtime. It cannot refer at compile time to objects that do not yet exist, and, for that reason, the one part of SQL that PL/ SQL does not include is DDL (Data Definition Language)—the statements, such as CREATE TABLE, that create the database and the objects it contains. However, you can work around this by using the package DBMS_SQL, included with the server, to generate the DDL code itself dynamically at runtime. For more information, see "Using DDL and Dynamic SQL" in the PL/SQL User's Guide and Reference.

Basic Structure and Syntax

PL/ SQL, like many programming languages, groups statements into units called blocks. These can either be named, in which case they are called subprograms, or unnamed, in which case they are anonymous blocks. Subprograms can be either functions or procedures. The difference between these, as in most languages, is that a function is used in an expression and returns a value to that expression, while a procedure is invoked as a standalone statement and passes values to the calling program only through parameters. Subprograms can be nested within one another and can be grouped in larger units called packages.

MAC003117

ORCL00941793

A block has three parts:

- A DECLARE section. This is where you define local variables, constants, types, exceptions, and nested subprograms. PL/ SQL has a forward declaration, but you can use it only for subprograms. Therefore, you must define all variables, constant, and types before referencing them. For more information on forward declarations, see "Declaring Subprograms" in the PL/SQL User's Guide and Reference.
- An EXECUTABLE section. This is the actual code that the block executes. This is the only part of the block that must always be present.
- An EXCEPTION section. This is a section for handling runtime errors and warnings.

These divisions are explained further in the sections that follow.

The DECLARE Section

The DECLARE section begins with the keyword DECLARE and ends when the keyword BEGIN signals the arrival of the EXECUTABLE section. You can declare types, constants, variables, exceptions, and cursors in any order, as long as they are declared before they are referenced in another definition. You declare subprograms last. A semi–colon terminates each definition.

Datatypes

PL/ SQL provides a number of predefined datatypes for variables and constants. It also enables you to define your own types, which are subtypes of the predefined types. The types fall into the following three categories:

- Scalar. These include all string, number, and binary types. All of the SQL datatypes, which are the datatypes that you can store in the database, fall into this category. To find out about these datatypes, see "Datatypes" in the Oracle7 Server SQL Reference.
- Composite. These are structured datatypes, which is to say data structures that have components you can address independently. The PL/ SQL composite types are TABLE (which is distinct from both database and HTML tables) and RECORD. These types are explained later in this chapter.
- Reference. There is one kind of reference datatype—REF CURSOR—which is a pointer to a cursor. Cursors are explained

MAC003118

ORCL00941794

later in this chapter. For more information on the REF CURSOR datatype, see "Using Cursor Variables" in the PL/SQL User's Guide and Reference.

For a list and explanation of all PL/SQL datatypes, see "Datatypes" in the PL/SQL User's Guide and Reference.

In many cases, you can convert from one datatype to another, either explicitly or automatically. The possible conversions and the procedure involved are explained in the PL/SQL User's Guide and Reference under "Datatype Conversion".

You can also define a variable so that it inherits its datatype from a database column or from another variable or constant, as explained in the next section.

Declaring Variables

For variables, provide the name, datatype, and any desired attributes, as follows:

```
cnum INTEGER(5) NOT NULL;
```

This declares a five–digit integer called cnum that will not accept nulls. The use of case above serves to distinguish keywords from identifiers; PL/SQL is not case–sensitive. NOT NULL is the only SQL constraint that you can use as a PL/SQL attribute.

> Note: PL/SQL initializes all variables to null. Therefore, a NOT NULL variable, such as the above, produces an error if referenced before it is assigned a value.

Optionally, you can assign an initial value to the variable when you declare it by following the datatype specification with an assignment, as follows:

```
cnum INTEGER(5) := 254;
```

This sets cnum to the initial value of 254. Alternatively, you can use the keyword DEFAULT in place of the assignment operator := to achieve the same effect. For more information on setting defaults, see "Declarations" in the PL/SQL User's Guide and Reference.

Inheriting Datatypes

To have the variable inherit the datatype of a database column or of another variable, use the %TYPE attribute in place of a declared datatype, as follows:

```
snum cnum%TYPE;
```

This means that snum inherits the datatype of cnum. You can inherit datatypes from database columns in the same way, by using the notation tablename.columnname in place of the variable name. Normally, you do this if the variable in question is to place values in or retrieve them from the column. The advantages are that you need not know the exact datatype the column uses and that you need not change your code if the datatype of that column changes. If you do not own the table containing the column, precede the tablename with the schemaname, as described under "Naming Conventions" elsewhere in this chapter. For more information on %TYPE assignments, see "Declarations" in the PL/SQL User's Guide and Reference.

Declaring Constants

You declare constants the same way as variables, except for the addition of the keyword CONSTANT and the mandatory assignment of a value. Constants do not take attributes other than the value. An example follows:

```
interest CONSTANT REAL(5,2) := 759.32;
```

Defining Types

User–defined types in PL/SQL are subtypes of existing datatypes. They provide you with the ability to rename types and to constrain them by specifying for your subtype lengths, maximum lengths, scales, or precisions, as appropriate to the standard datatype on which the subtype is based. For more information on the datatype parameters, see "Datatypes" in Chapter 2 of the Oracle7 Server SQL Reference. For more information on PL/SQL datatypes, see "Datatypes" in the PL/SQL User's Guide and Reference. You can also use the %TYPE attribute in defining a subtype. Here is an example:

```
SUBTYPE shortnum IS INTEGER(3);
```

This defines SHORTNUM as a 3–digit version of INTEGER. For more information see "User–Defined Subtypes" in the PL/SQL User's Guide and Reference.

Scope and Visibility

Nested subprograms, defined in the DECLARE section, can be called from either of the other sections, but only from within the same block where they are defined or within blocks contained in that block. Variables, constants, types, and subprograms defined within a block are local to that block, and their definitions are not meaningful outside of it. Objects that are local to a block may be used by subprograms

MAC003120

ORCL00941796

contained at any level of nesting in that same block. Such objects are global to the block that calls them.

The area of a program within which an object can be used is called the object's scope. An object's scope is distinct from its visibility. The former is the area of the program that can reference the object; the latter is the, generally smaller, portion that can reference it without qualification.

An ambiguous reference can arise because objects or subprograms contained in different blocks can have the same names, even if they have overlapping scopes. When this happens, the reference by default means the object most local in scope—in other words, the first one PL/ SQL finds by starting in the current block and working out to the enclosing ones. Qualification is the method used to override this. It is similar to the system of qualification used for database objects, as explained under "Naming Conventions" elsewhere in the chapter. To qualify an object's name, precede it with the name of the subprogram where it is declared, followed by a dot, as follows:

```
relocate.transmit(245, destination);
```

This invokes a procedure called transmit declared in some subprogram called relocate. The subprogram relocate must be global to the block from which it is called.

Data Structures

PL/ SQL provides two structured datatypes: TABLE and RECORD. It also provides a data structure called a cursor that holds the results of queries. Cursors are different from the other two in that you declare variables and constants to be of type TABLE or RECORD just as you would any other datatype. Cursors, on the other hand, have their own syntax and their own operations. Explanations of these types follow:

PL/SQL Tables

These are somewhat similar to database tables, except that they always consist of two columns: a column of values and a primary key. This also makes them similar to one–dimensional arrays, with the primary key functioning as the array index. Like SQL tables, PL/ SQL tables have no fixed allocation of rows, but grow dynamically. One of their main uses is to enable you to pass entire columns of values as parameters to subprograms. With a set of such parameters, you can pass an entire table. The primary key is always of type BINARY_INTEGER, and the values can be of any scalar type.

You declare objects of type TABLE in two stages:

MAC003121

ORCL00941797

1. You declare a subtype using the following syntax:

```
TYPE type_name IS TABLE OF
    datatype_spec
    [ NOT NULL ]
    INDEX BY BINARY INTEGER;
```

Where datatype_spec means the following:

```
datatype | variablename%TYPE | tablename.columnname%TYPE
```

In other words, you can either specify the type of values directly or use the %TYPE attribute (explained under "Declaring Variables", elsewhere in this chapter) to inherit the datatype from an existing variable or database column.

2. You assign objects to this subtype in the usual way. You cannot assign initial values to tables, so the first reference to the table in the EXECUTABLE section must provide it at least one value.

When you reference PL/SQL tables, you use an array–like syntax of the form:

```
column_value(primary_key_value)
```

In other words, the third row (value) of a table called "Employees" would be referenced as follows:

```
Employees(3)
```

You can use these as ordinary expressions. For example, to assign a value to a table row, use the following syntax:

```
Employees(3) := 'Marsha';
```

For more information, see "PL/SQL Tables" in the PL/SQL User's Guide and Reference.

Records

As in many languages, these are data structures that contain one or more fields. Each record of a given type contains the same group of fields with different values. Each field has a datatype, which can be RECORD. In other words, you can nest records, creating data structures of arbitrary complexity. As with tables, you declare records by first declaring a subtype, using the following syntax:

```
TYPE record_type IS RECORD
    (fieldname datatype[, fieldname datatype]...);
```

The second line of the above indicates a parenthesized, comma–separated, list of fieldnames followed by datatype specifications. The datatype specifications can be direct or be inherited

MAC003122

ORCL00941798

using the %TYPE attribute, as shown for TABLE and as explained under "Declaring Variables", elsewhere in this chapter.

You can also define a record type that automatically mirrors the structure of a database table or of a cursor, so that each record of the type corresponds to a row, and each field in the record corresponds to a column. To do this, use the %ROWTYPE attribute with a table or cursor name in the same way you would the %TYPE attribute with a variable, or column. The fields of the record inherit the column names and datatypes from the cursor or table. For more information, see "Records" and "%ROWTYPE Attribute" in the PL/SQL User's Guide and Reference.

Cursors

A cursor is a data structure that holds the results of a query (a SELECT *statement) for processing by other statements. Since the output of any* query has the structure of a table, you can think of a cursor as a temporary table whose content is the output of the query.

When you declare a cursor, you associate it with the desired query. When you want to use that cursor, you open it, executing the associated query and filling the cursor with its results. You then fetch each row of the query's output in turn for processing by other statements in the program. You can also use a cursor to update a table's contents. To do this, use a FOR UPDATE clause to lock the rows in the table. See "Using FOR UPDATE" in the PL/SQL User's Guide and *Reference for more information. Sometimes, you may need to use cursor* variables, which are not associated with a query until runtime. This is a form of dynamic SQL.

For more information on cursor variables, see "Using Dynamic SQL" in the Oracle7 Server Application Developers Guide and "Cursor Variables" in the PL/SQL User's Guide and Reference.

For more information on cursors in general, see "Cursors" in the PL/SQL User's Guide and Reference. See also "DECLARE CURSOR," "OPEN", and "FETCH" in the Oracle7 Server SQL Reference.

You can simplify some cursor operations by using cursor FOR loops. For more information on these, see "Using Cursor FOR Loops" in the PL/SQL User's Guide and Reference.

Exceptions

You also use the DECLARE section to define your own error conditions, called "exceptions". Explanation of this is deferred until the "EXCEPTION Section" portion of this chapter.

MAC003123

ORCL00941799

Declaring Subprograms

You must place all subprogram declarations at the end of the declare section, following all variable, constant, type, and exception declarations for the block. The syntax is as follows:

```
PROCEDURE procedure_name (parameter_name datatype, parameter_name
datatype...) IS
            {local declarations}
    BEGIN {executable code}
    EXCEPTION
    END;
```

> Note: For subprograms, the keyword DECLARE is omitted before the local declarations. Place local declarations before the keyword BEGIN, as shown.

The names you give the parameters in the declaration are the names that the procedure itself uses to refer to them. These are called the formal parameters. When the procedure is invoked, different variables or constants may be used to pass values to or from the formal parameters; these are called the actual parameters.

When calling the procedure, you can use each parameter for input of a value to the procedure, output of a value from it, or both. These correspond to the three parameter modes: IN, OUT, and IN/ OUT. For more information, see "Parameter Modes" in the PL/SQL User's Guide and Reference.

When you call the procedure, you can match the actual to the formal parameters either implicitly, by passing them in the same order they are given in the declaration, or explicitly, by naming the formal followed by the actual parameter as shown:

```
transmit(destination => address);
```

This invokes a procedure called transmit, assigning the value of address as the actual parameter for the formal parameter destination. This implies that the parameter destination is used within the transmit procedure and that the parameter address is used outside of it. Usually, it is good programming practice to use different names for matching formal and actual parameters. For more information on this, see "Positional and Named Notation" in the PL/SQL User's Guide and Reference.

Functions are the same, except for the addition of a return value, specified as follows:

```
FUNCTION function_name (parameter_name, parameter_name
datatype...)
    RETURN datatype IS
```

```
{local declarations}
BEGIN {executable code}
EXCEPTION {local exception handlers}
END;
```

Again, line breaks are only for readability. A RETURN statement in the executable section actually determines what the return value is. This consists of the keyword RETURN followed by an expression. When the function executes the RETURN statement, it terminates and passes the value of that expression to whichever statement called it in the containing block.

You can also use the RETURN statement without an expression in a procedure to force the procedure to exit.

For more information on procedures and functions, see "Declaring Subprograms" in the PL/SQL User's Guide and Reference.

The EXECUTABLE Section

The executable section is the main body of code. It consists primarily of SQL statements, flow control statements, and assignments. SQL statements are explained earlier in this chapter; assignments and flow–control statements are explained in the sections that follow.

Assignments

The assignment operator is :=. For example, the following statement assigns the value 45 to the variable a:

```
a := 45;
```

Character strings should be set off with single quotes (') as in all expressions. An example follows:

```
FNAME := 'Clair';
```

There are other examples of assignments in other parts of this chapter.

Flow Control

PL/SQL supports the following kinds of flow–control statements:

- IF statements. These execute a group of one or more statements based on whether a condition is TRUE.

MAC003125

ORCL00941801

- Basic loops. These repeatedly execute a group of one or more statements until an EXIT statement is reached.
- FOR loops. These repeatedly execute a group of one or more statements a given number of times or until an EXIT statement is reached.
- WHILE loops. These repeatedly execute a group of one or more statements until a particular condition is met or an EXIT statement is reached.
- GOTO statements. These pass execution directly to another point in the code, exiting loops and enclosing blocks as necessary. Use these sparsely, as they make code difficult to read and debug.

If you know other programming languages, you probably are familiar with most or all of these types of statements. The following sections describe the PL/ SQL versions of them in greater detail. For more information on any of these, see "Control Structures" in the PL/SQL User's Guide and Reference.

You can nest flow control statements within one another to any level of complexity.

IF Statements

These are similar to the IF statement in many other languages, except that they use predicates, which are three–valued Boolean expressions like the SQL predicates discussed earlier in this chapter. In most respects, a Boolean NULL behaves like a Boolean FALSE, except that negation does not make it positive, but leaves it NULL.

The IF statement has the following forms:

```
1   IF <condition> THEN <statement–list>;
    END IF;
```

If the condition following IF is TRUE, PL/ SQL executes the statements in the list following THEN. A semicolon terminates this list. END IF (not ENDIF) is mandatory and terminates the entire IF statement. Here is an example:

```
IF balance > 500 THEN send_bill(customer);
END IF;
```

We are assuming that send_bill is a procedure taking a single parameter.

```
2   IF <condition> THEN <statement–list>;
    ELSE <statement–list>;
    END IF;
```


MAC003126

ORCL00941802

This is the same as the preceding statement, except that, if that condition is FALSE or NULL, PL/ SQL executes the statement list following ELSE instead of that following THEN.

```
3   IF <condition> THEN <statement-list>;
    ELSIF <condition> THEN <statement-list>;
    ELSIF <condition> THEN <statement-list>; . . . . .
    ELSE <statement-list>;
    END IF;
```

You can include any number of ELSIF (not ELSEIF) conditions. Each is tested only if the IF condition and all preceding ELSIF conditions are FALSE or NULL. As soon as PL/ SQL finds an IF or ELSIF condition that is TRUE, it executes the associated THEN statement list and skips ahead to END IF. The ELSE clause is optional, but, if included, must come last. It is executed if all preceding IF and ELSIF conditions are FALSE or NULL.

NULL Statements

If you do not want an action to be taken for a given condition, you can use the NULL statement, which is not to be confused with database nulls, Boolean NULLs, or the SQL predicate IS NULL. The syntax of this statement is simply:

```
NULL;
```

The statement performs no action, but fulfills the syntax requirement that a statement list must follow every THEN keyword. In some cases, you can also use it to increase the readability of your code. For more information on the NULL statement, see "NULL Statement" in the PL/SQL User's Guide and Reference.

Basic Loops

A basic loop is a loop that keeps repeating until an EXIT statement is reached. The EXIT statement must be within the loop itself. If no EXIT (or GOTO) statement ever executes, the loop is infinite. An example follows:

```
credit := 0;
 LOOP
    IF c = 5 THEN EXIT;
    END IF;
    credit := credit + 1;
END LOOP;
```


MAC003127

ORCL00941803

This loop keeps incrementing credit until it reaches 5 and then exits. An alternative to placing an exit statement inside an IF statement is to use the EXIT–WHEN syntax, as follows:

```
EXIT WHEN credit = 5;
```

This is equivalent to the earlier IF statement.

> Note: The EXIT statement cannot be the last statement in a PL/ SQL block. If you want to exit a PL/ SQL block before its normal end is reached, use the RETURN statement. For more information, see "RETURN Statement" in the PL/SQL User's Guide and Reference.

FOR Loops

A FOR loop, as in most languages, repeats a group of statements a given number of times. The following FOR loop is equivalent to the example used for basic loops, except that it also changes a variable called interest.

```
FOR credit IN 1..5 LOOP
    interest := interest * 1.2;
END LOOP;
```

The numbers used to specify the range (in this case, 1 and 5) can be variables, so you can let the number of iterations of the loop be determined at runtime if you wish.

WHILE Loops

A WHILE loop repeats a group of statements until a condition is met. Here is a WHILE loop that is the equivalent of the preceding example:

```
credit := 1;
WHILE credit <= 5 LOOP
    interest := interest * 1.2;
    credit := credit + 1;
END LOOP;
```

Unlike some languages, PL/ SQL has no structure, such as REPEAT–UNTIL, that forces a LOOP to execute at least once. You can create this effect, however, using either basic or WHILE loops and setting a variable to a value that will trigger the loop, as in the above example. For more information on loops, see "Iterative Control" in the PL/SQL User's Guide and Reference.

MAC003128

ORCL00941804

GOTO Statements

A GOTO statement immediately transfers execution to another point in the program. The point in the program where the statement is to arrive must be preceded by a label. A label is an identifier for a location in the code. It must be unique within its scope and must be enclosed in double angle brackets, as follows:

```
<<this_is_a_label>>
```

You only use the brackets at the target itself, not in the GOTO statement that references it, so a GOTO statement transferring execution to the above label would be:

```
GOTO this_is_a_label;
```

> Note: An EXIT statement can also take a label, if that label indicates the beginning of a loop enclosing the EXIT statement. You can use this to exit several nested loops at once. See "Loop Labels" in the PL/SQL User's Guide and Reference for more information.

A GOTO statement is subject to the following restrictions:

- It must branch to an executable statement, not, for example, an END.
- It cannot branch to a point within the body of IF or a LOOP statement, unless it is contained in the body of that statement itself.
- It cannot branch to a subprogram or enclosing block of the present block (with one exception, explained shortly).
- It cannot branch from one IF statement clause to another. That is to say, it cannot jump between THEN, ELSIF, and ELSE clauses that are part of the same IF statement.
- It cannot branch from the EXCEPTION section to the EXECUTABLE section of the same block.
- It can, however, branch from the EXCEPTION section of a block to the EXECUTABLE section of an enclosing block, which is the exception to the third rule above.

The EXCEPTION Section

The EXCEPTION section follows the END that matches the BEGIN of the EXECUTABLE section and begins with the keyword EXCEPTION.

MAC003129

ORCL00941805

It contains code that responds to runtime errors. An exception is a specific kind of runtime error. When that kind of error occurs, you say that the exception is raised. An exception handler is a body of code designed to handle a particular exception or group of exceptions. Exception handlers, like the rest of the code, are operative only once the code is compiled and therefore can do nothing about compilation errors.

There are two basic kinds of exceptions: predefined and user–defined. The predefined exceptions are provided by PL/ SQL in a package called STANDARD. They correspond to various runtime problems that are known to arise often—for example, dividing by zero or running out of memory. These are listed in the PL/SQL User's Guide and Reference under "Predefined Exceptions".

The Oracle Server can distinguish between and track many more kinds of errors than the limited set that STANDARD predefines. Each of Oracle's hundreds of messages are identified with a number, and STANDARD has simply provided labels for a few of the common ones. You can deal with the other messages in either or both of two ways:

- You can define your own exception labels for specified Oracle messages using a pragma (a compiler directive). This procedure will be explained shortly.
- You can define a handler for the default exception OTHERS. Within that handler, you can identify the specific error by accessing the built–in functions SQLCODE and SQLERRM, which contain, respectively, the numeric code and a prose description of the message.

You can also define your own exceptions as will be shown. It is usually better, however, to use Oracle exceptions where possible, because then the conditions are tested automatically when each statement is executed, and an exception is raised if the error occurs.

Declaring Exceptions

PL/ SQL predefined exceptions, of course, need not be declared. You declare user–defined exceptions or user–defined labels for Oracle messages in the DECLARE section, similarly to variables. An example follows:

```
customer_deceased  EXCEPTION;
```

In other words, an identifier you choose followed by the keyword EXCEPTION. Notice that all this declaration has done is provide a name. The program still has no idea when this exception should be

MAC003130

ORCL00941806

raised. In fact, there is at this point no way of telling if this is to be a user–defined exception or simply a label for an Oracle message.

Labeling Oracle Messages

If a previously–declared exception is to be a label for an Oracle error, you must define it as such with a second statement in the DECLARE section, as follows:

```
PRAGMA EXCEPTION_INIT (exception_name, Oracle_error_number);
```

A PRAGMA is a instruction for the compiler, and EXCEPTION_INIT is the type of PRAGMA. This tells the compiler to associate the given exception name with the given Oracle error number. This is the same number to which SQLCODE is set when the error occurs. The advantage of this over defining your own error condition is that you pass the responsibility for determining when the error has occurred and raising the exception to Oracle. You can find the numeric codes and explanations for Oracle messages in Oracle7 Server Messages.

User–Defined Exceptions

If the declared condition is not to be a label for an Oracle error, but a user–defined error, you do not need to put another statement referring to it in the DECLARE section. In the EXECUTABLE section, however, you must test the situation you intend the exception to handle whenever appropriate and raise the condition manually, if needed. Here is an example:

```
IF cnum < 0 THEN RAISE customer_deceased;
```

You can also use the RAISE statement to force the raising of predefined exceptions. For more information, see "Error Handling" in the PL/SQL User's Guide and Reference.

Handling Exceptions

Once an exception is raised, whether explicitly with a RAISE statement or automatically by Oracle, execution passes to the EXCEPTION section of the block, where the various exception handlers reside. If a handler for the raised exception is not found in the current block, enclosing blocks are searched until one is found. If PL/ SQL finds an OTHERS handler in any block, execution passes to that handler. An OTHERS handler must be the last handler in its block. If no handler for an exception is found, Oracle raises an unhandled exception error. Note: this does not automatically roll back (undo) changes made by the

subprogram, which might leave the database in an undesirable intermediate state.

This is the syntax of an exception handler:

```
WHEN exception_condition THEN statement_list;
```

The exception is the identifier for the raised condition. If desired, you can specify multiple exceptions for the same handler, separated by the keyword OR. The exception can be either one the package STANDARD provided or one you declared. The statement list does what is appropriate to handle the error—writing information about it to a file, for example—and arranges to exit the block gracefully if possible. Although exceptions do not necessarily force program termination, they do force the program to exit the current block. You cannot override this with a GOTO statement. You can use a GOTO within an exception handler, but only if its destination is some enclosing block.

> Note: If you have an error prone statement and want execution to continue following this statement, even when an exception occurs, put the statement, including the appropriate exception handlers, in its own block, so that the current block becomes the enclosing block.
>
> Note: If an exception occurs in the DECLARE section or the EXCEPTION section itself, local exception handlers cannot address it; execution passes automatically to the EXCEPTION section of the enclosing block.

Storing Procedures and Functions in the Database

To have a procedure or function stored as a database object, you issue a CREATE PROCEDURE or a CREATE FUNCTION statement directly to the server using SQL*PLUS or Server Manager. The easy way to do this is to use your ordinary text editor to produce the CREATE statement and then to load it as a script. This process is explained under "Creating Stored Procedures and Functions" in the Oracle7 Server Application Developers Guide. This approach is recommended because you often create entire groups of procedures and functions together. These groups are called "packages" and are explained later in this chapter.

The syntax for these statements is slightly different than that used to declare subprograms in PL/ SQL, as the following example shows:

```
CREATE PROCEDURE fire_employee (empno INTEGER) IS
    BEGIN
```


MAC003132

ORCL00941808

```
    DELETE FROM Employees WHERE enum = empno;
END;
```

As you can see, the main difference is the addition of the keyword CREATE. You also have the option of replacing the keyword IS with AS, which does not affect the meaning. To replace an existing procedure of the same name with this procedure (as you frequently may need to do during development and testing), you can use CREATE OR REPLACE instead of simply CREATE. This destroys the old version, if any, without warning.

Privileges Required

A stored procedure or function (for the rest of this discussion, "procedure" shall mean "procedure or function" unless otherwise indicated or clear from context) is a database object like a table. It resides in a schema, and its use is controlled by privileges. To create a procedure and have it compile successfully, you must meet the following conditions:

- If the procedure is to be in your own schema, you must have the CREATE PROCEDURE or the CREATE ANY PROCEDURE system privilege. These privileges apply as well to functions.
- If the procedure is to be in a schema you do not own, you must have the CREATE ANY PROCEDURE system privilege.
- You must have the object privileges necessary to perform all operations contained in the procedure. You must have these privileges as a user, not through roles. If your privileges change after you have created the procedure, the procedure may no longer be executable.

To enable others to use the procedure, grant them the EXECUTE privilege on it using the SQL statement GRANT (see "GRANT" in Chapter 4 of the Oracle7 Server SQL Reference). When these users execute the procedure, they do so under your privileges, not their own. Therefore, you do not have to grant them the privileges to perform these actions outside the control of the procedure, which is a useful security feature. To enable all users to use the procedure, grant EXECUTE to PUBLIC. The following example permits all users to execute a procedure called show_product.

```
GRANT EXECUTE ON show_product TO PUBLIC;
```

Of course, the public normally does not execute such a procedure directly. This statement enables you to use the procedure in your PL/SQL code that is to be publicly executable. If multiple users access

the same procedure simultaneously, each gets his own instance. This means that the setting of variables and other activities by different users do not affect one another.

For more information on privileges and roles, see "GRANT" in Chapter 4 of the Oracle7 Server SQL Reference. There are three versions of GRANT listed—one each for object privileges, system privileges, and roles.

For more information on storing procedures and functions in the database, see "Storing Procedures and Functions" in the Oracle7 Server Application Developers Guide and see "CREATE FUNCTION" and "CREATE PROCEDURE" in the Oracle7 Server SQL Reference.

Packages

A package is a group of related PL/SQL objects (variables, constants, types, and cursors) and subprograms that is stored in the database as a unit. Being a database object, a package resides in a schema, and its use is controlled by privileges. Among its differences from regular PL/SQL programs are that a package as such does not do anything. It is a collection of subprograms and objects, at least some of which are accessible to applications outside of it. It is the subprograms in the package that contain the executable code. A package has the following two parts:

- The package specification is the public interface to the package. It declares all objects and subprograms that are to be accessible from outside the package. Packages do not take parameters, so these constitute the entire public interface.
- The package body is the internal portion of the package. It contains all objects and subprograms that are to be local to the package. It also contains definitions of the public cursors and subprograms. The package specification declares but does not define these.

One of the advantages of using packages is that the package specification is independent of the body. You can change the body and, so long as it still matches the specification, no changes to other code are needed, nor will any other references become invalid.

Packages cannot be nested, but they can call one another's public subprograms and reference one another's public objects.

MAC003134

ORCL00941810

Instantiation of Packages

It is important to realize that a package is instantiated once for a given user session. That is to say, the values of all variables and constants, as well as the contents and state of all cursors, in a package, once set, persist for the duration of the session, even if you exit the package. When you reenter the package, these objects retain the values and state they had before, unless they are explicitly reinitialized. Of course, another user has another session and therefore another set of values. Nonetheless, a global reinitialization of a package's objects for you does not take place until you disconnect from the database.

There is an exception, however. When one package calls another, execution of the second has a dependency on the first. If the first is invalidated, for example because its creator loses a privilege that the package requires, the second, while not necessarily invalidated, becomes deinstantiated. That is to say, all its objects are reinitialized.

> Note: In PL/ SQL, stored procedures and packages are automatically recompiled if changes to the database mandate it. For example, a change to the datatype of a column can automatically cascade to a variable referencing that column if the former is declared with the %TYPE attribute, but that change requires that the PL/ SQL procedure declaring that variable be recompiled. So long as the PL/ SL code as written is still valid, the recompilation occurs automatically and invisibly to the user.

Creating Packages

To create a package, you use the SQL statement CREATE PACKAGE for the specification and CREATE PACKAGE BODY for the body. You must create the specification first. Sometimes, a package may consist of only public variables, types, and constants, in which case no body is necessary. Generally, however, you use both parts.

> Note: Before you can create a package, the special user SYS must run the SQL script DBMSSTDX.SQL. The exact name and location of this script may vary according to your operating system. Contact your database administrator if you are not sure this script has been run.

Creating the Package Specification

The syntax of the CREATE PACKAGE statement is as follows:

MAC003135

ORCL00941811

```
CREATE [OR REPLACE] PACKAGE package_name IS
    {PL/SQL declarations}
    END;
```

The optional OR REPLACE clause operates just as it does for stored procedures, as explained elsewhere in this chapter. The PL/SQL declarations are as outlined under DECLARE SECTION elsewhere in this chapter, except that the keyword DECLARE is not used and that the subprogram and cursor declarations are incomplete. For subprograms, you provide only the name, parameters, and, in the case of functions, the datatype of the return value. For cursors, provide the name and a new item called the return type. This approach hides the implementation of these objects from the public while making the objects themselves accessible.

The syntax for declaring a cursor with a return type is as follows:

```
CURSOR c1 IS RETURN return_type;
```

The return type is always some sort of record type that provides a description of the cursor's output. The structure of this record is to mirror the structure of the cursor's rows. You can specify it using any of the following:

- A record subtype previously defined and in scope. For more information, see "Records" elsewhere in this chapter.
- A type inherited from such a record subtype using the %TYPE attribute. For more information, see "Declaring Variables" elsewhere in this chapter.
- A type inherited from a table, most likely the table the cursor queries, using the %ROWTYPE attribute. For more information, see "Records" elsewhere in this chapter.
- A type inherited from a cursor using the %ROWTYPE attribute. For more information, see "Records" elsewhere in this chapter.

For more information, see CREATE PACKAGE in Chapter 4 of the Oracle7 Server SQL Reference, "Packages" in the PL/SQL User's Guide and Reference, and "Using Procedures and Packages" in the Oracle7 Server Application Developer's Guide.

Creating the Package Body

To create the package body, use the CREATE PACKAGE BODY statement. The syntax is as follows:

```
CREATE [OR REPLACE] PACKAGE BODY package_name IS
    {PL/SQL declarations}
    END;
```


MAC003136

ORCL00941812

Since a package as such does not do anything, the PL/ SQL code still consists only of a DECLARE section with the keyword DECLARE omitted. It is the subprograms within the package that contain the executable code. Variables, constants, types, and cursors declared directly (in other words, not within a subprogram) in the declare section have a global scope within the package body. Variables, constants, and types already declared in the package specification are public and should not be declared again here.

Public cursors and subprograms, however, must be declared again here, as their declarations in the specification is incomplete. This time the declarations must include the PL/ SQL code (in the case of subprograms) or the query (in the case of cursors) that is to be executed. For subprograms, the parameter list must match that given in the package specification word for word (except for differences in white space). This means, for example, that you cannot specify a datatype directly in the specification and use the %TYPE attribute to specify it in the body.

You can create an initialization section at the end of the package body. This is a body of executable code—chiefly assignments—enclosed with the keywords BEGIN and END. Use this to initialize constants and variables that are global to the package, since otherwise they could be initialized only within subprograms, and you have no control of the order in which subprograms are called by outside applications. This initialization is performed only once per session.

For more information, see CREATE PACKAGE BODY in the Oracle7 Server SQL Reference, "Packages" in the PL/SQL User's Guide and Reference, and "Using Procedures and Packages" in the Oracle7 Server Application Developers Guide.

Overloading Subprograms

Within a package, subprogram names need not be unique, even at the same level of scope. There can be multiple like–named subprograms in the same declare section, provided that the parameters that they take differ in number, order, or datatype and that, when the procedures are called, the values passed by the calling procedure (the actual parameters) match or can be automatically converted to the datatypes specified in the declaration (the formal parameters). To find out which datatypes PL/ SQL can convert automatically, look under "Datatype Conversion" in the PL/SQL User's Guide and Reference.

The reason this is permitted is so you can overload subprograms. Overloading permits you to have several versions of a procedure that are conceptually similar but behave differently with different

MAC003137

ORCL00941813

parameters. This is one of the properties of object–oriented programming. For more information on overloading, see "Overloading" in the PL/SQL User's Guide and Reference.

Database Triggers

Triggers are blocks of PL/ SQL code that execute automatically in response to events. Database triggers reside in the database and respond to changes in the data. They are not to be confused with application triggers, which reside in applications and are beyond the scope of this discussion. Database triggers are a technology that for the most part has superceded application triggers.

You create triggers as you do stored procedures and packages, by using your text editor to write scripts that create them and then using SQL*Plus or Server Manager to run these scripts. A trigger is like a package in that:

- It takes no parameters as such. It refers to, responds to, and possibly affects the data in the database.
- It cannot be directly called like a procedure. To fire (execute) a trigger, you must make the database change to which it responds. If you only want to test the trigger, you can rollback (undo) the database change that you made after the trigger fires.

Triggers can be classified in three ways:

- INSERT triggers, UPDATE triggers, and DELETE triggers. This is a classification based on the statement to which the trigger responds. The categories are not mutually exclusive, meaning one trigger can respond to any or all of these statements.
- Row triggers and statement triggers. Any of the above statements can affect any number of rows in a table at once. A row trigger is fired once for each row affected. A statement trigger is fired once for each statement, however many rows it affects.
- BEFORE triggers and AFTER triggers. This specifies whether the trigger is fired before or after the data modification occurs.

As you can see, all three of these classifications apply to all triggers, so that there are, for example, BEFORE DELETE OR INSERT statement triggers and AFTER UPDATE row triggers.

MAC003138

ORCL00941814

Creating Triggers

The syntax of the CREATE TRIGGER statement is as follows:

```
CREATE [OR REPLACE] TRIGGER trigger_name
    BEFORE | AFTER
    DELETE | INSERT | UPDATE [OF column_list]
    ON table_name
    [ FOR EACH ROW[ WHEN predicate ] ]
    {PL/SQL block};
```

In the above, square brackets ([ ]) enclose optional elements. Vertical bars (|) indicate that what precedes may be replaced by what follows.

In other words, you must specify the following:

- A trigger name. This is used to alter or drop the trigger. The trigger name must be unique within the schema.
- BEFORE or AFTER. This specifies whether this is a BEFORE or AFTER trigger.
- INSERT, UPDATE, or DELETE. This specifies the type of statement that fires the trigger. If it is UPDATE, you optionally can specify a list of one or more columns, and only updates to those columns fire the trigger. In such a list, separate the column names with commas and spaces. You may specify this clause more than once for triggers that are to respond to multiple statements; if you do, separate the occurrences with the keyword OR surrounded by white space.
- ON table_name. This identifies the table with which the trigger is associated.
- PL/ SQL Block. This is an anonymous PL/ SQL block containing the code the trigger executes.

You optionally can specify the following:

- OR REPLACE. This has the usual effect.
- FOR EACH ROW [WHEN predicate]. This identifies the trigger as a row trigger. If omitted, the trigger is a statement trigger. Even if this clause is included, the WHEN clause remains optional. The WHEN clause contains a SQL (not a PL/ SQL) predicate that is tested against each row the triggering statement alters. If the values in that row make the predicate TRUE, the trigger is fired; else it is not. If the WHEN clause is omitted, the trigger is fired for each altered row.

Here is an example:

MAC003139

ORCL00941815

```
CREATE TRIGGER give_bonus
    AFTER UPDATE OF sales
    ON salespeople
    FOR EACH ROW WHEN sales > 8000.00
    BEGIN
    UPDATE salescommissions SET bonus = bonus + 150.00;
    END;
```

This creates a row trigger called give_bonus. Every time the sales column of the salespeople table is updated, the trigger checks to see if it is over 8000.00. If so, it executes the PL/SQL block, consisting in this case of a single SQL statement that increments the bonus column in the salescommissions table by 150.00.

Privileges Required

To create a trigger in your own schema, you must have the CREATE TRIGGER system privilege and one of the following must be true:

- You own the table associated with the trigger.
- You have the ALTER privilege on the table associated with the trigger.
- You have the ALTER ANY TABLE system privilege.

To create a trigger in another user's schema, you must have the CREATE ANY TRIGGER system privilege. To create such a trigger, you precede the trigger name in the CREATE TRIGGER statement with the name of the schema wherein it will reside, using the conventional dot notation.

Referring to Altered and Unaltered States

You can use the correlation variables OLD and NEW in the PL/SQL block to refer to values in the table before and after the triggering statement had its effect. Simply precede the column names with these variables using the dot notation.

If these names are not suitable, you can define others using the REFERENCING clause of the CREATE TRIGGER statement, which is omitted from the syntax diagram above for the sake of simplicity. For more information on this clause, see CREATE TRIGGER in the Oracle7 Server SQL Reference.

Note: if a trigger raises an unhandled exception, its execution fails and the statement that triggered it is rolled back if necessary. This enables you to use triggers to define complex constraints. If the effects of the trigger have caused a change in the value of package body variables, however, this change is not reversed. You should try to design your

MAC003140

ORCL00941816

packages to spot this eventuality. For more information, see "Using Database Triggers" in the Oracle7 Server Application Developers Guide.

Enabling and Disabling Triggers

Just because a trigger exists does not mean it is in effect. If the trigger is disabled, it does not fire. By default, all triggers are enabled when created, but you can disable a trigger using the ALTER TRIGGER statement. To do this, the trigger must be in your schema, or you must have the ALTER ANY TRIGGER system privilege. Here is the syntax:

```
ALTER TRIGGER trigger_name DISABLE;
```

Later you can enable the trigger again by issuing the same statement with ENABLE in place of DISABLE. The ALTER TRIGGER statement does not alter the trigger in any other way. To do that you must replace the trigger with a new version using CREATE OR REPLACE TRIGGER. For more information on enabling triggers, see ALTER TRIGGER in the *Oracle7 Server SQL Reference*.

For more information on triggers generally, see "Using Database Triggers" in the Oracle7 Server Application Developer's Guide and CREATE TRIGGER and DROP TRIGGER in the Oracle7 Server SQL Reference.

MAC003141

ORCL00941817

MAC003142

ORCL00941818

CHAPTER

5

Oracle Web Agent

The Oracle Web Agent uses the PL/ SQL language to integrate the Oracle7 Server and the World Wide Web. It provides transparent access to Oracle7 directly from the World Wide Web (hereafter Web).

The Oracle Web Agent provides an easy–to–use environment for building applications in PL/ SQL. These applications are then stored in the Oracle7 database. A programmer who has chosen PL/ SQL as the language in which to implement a CGI application need only worry about implementing the logic specific to the application itself, because the Oracle Web Agent provides utilities which take care of the repetitive tasks associated with deploying a CGI application.

The HTML output generated by a PL/ SQL procedure is sent from the Oracle7 Server to the Oracle Web Agent. The output is then passed on to the Oracle Web Listener. The Web Listener sends the generated HTML document back to the Web client.

The HTML administrative forms that simplify the task of configuring the Oracle Web Agent are documented in Chapter 7, "Oracle WebServer Administration Utility."

MAC003143

ORCL00941819

Components of the Oracle Web Agent

The Oracle Web Agent is a CGI program that the Web Listener executes when a request is received for a dynamic document. This program logs into the database and executes stored PL/ SQL procedures that have been specified as part of the URL.

To facilitate the development of PL/ SQL for the Web Agent, a WebServer Developer's Toolkit is provided. This toolkit includes the following packages:

- HTP (hypertext procedures) and HTF (hypertext functions) ease the generation of HTML tags from within the user's PL/ SQL code. HTP and HTF automatically generate the HTML tags retrieved by the Oracle Web Agent.
- OWA_UTIL contains utility functions to enable the programmer to more rapidly build HTML output.

Note: The Web Agent also includes a PL/ SQL package called OWA that contains utility functions called by the Web Agent to do such things as set up the CGI environment variables and extract the output generated by the PL/ SQL code from a PL/ SQL table. A PL/ SQL developer does not make calls to the OWA package directly, although it can be called from a URL and, therefore, directly by a user..

For detailed information about the Oracle WebServer Developer's Toolkit, see Chapter 6. For information on the WebServer Administration Utility for administering the Web Agent, see Chapter 7, " Oracle WebServer Administration Utility."

Who Uses the Oracle Web Agent

The Oracle Web Agent can be used by:

- A programmer who wants to develop Web applications that interact with Oracle7. The Oracle Web Agent enables the programmer to create PL/ SQL procedures that perform database operations and format the results into HTML documents. These documents then get returned to the Web client.

  Therefore, the programmer needs to know the following:

  – the hypertext functions and hypertext procedures that can be used to generate an HTML–formatted document. These are described in Chapter 6, " The WebServer Developer's Toolkit."

MAC003144

ORCL00941820

- the basics of the PL/ SQL procedural language. These are contained in Chapter 4, "Introduction to the Oracle7 Server and to PL/ SQL".

In addition, the following is useful for the programmer:

- which Common Gateway Interface environment variables are available to the PL/ SQL application, and how to access them (This is covered in this chapter.).
- how the Oracle Web Agent is invoked (This is covered in this chapter.)
- how the Web Agent handles parameters (covered in this chapter.)
- how to use the OWA_UTIL PL/ SQL Utility Package (covered in Chapter 6, "The WebServer Developer's Toolkit".

• An administrator who wants to install or set up the Oracle Web Agent, and use the Administration Utility to maintain the Web Agent.

The administrator needs to know the following:

- how to create, modify, and delete Web Agent services
- where to install the Developer's Toolkit PL/ SQL packages provided by the Web Agent

The following information is useful for the administrator:

- how the Oracle Web Agent works (covered in this chapter.)
- how the Web Agent handles errors (covered in this chapter.)

How the Oracle Web Agent Works

The Oracle Web Agent enables you to create PL/ SQL stored procedures that can access data in Oracle7, dynamically create an HTML document from that data using the Developer's Toolkit, and return that document to the client.

This is a step by step description of how the Oracle Web Agent works:

1. The user submits a URL from the browser.

   The user submits the URL by one of the following methods:

   - by selecting a link from an HTML page
   - by specifying a URL in the Open URL box of their browser

MAC003145

ORCL00941821

– by submitting an HTML form

This action invokes a GET or POST request method operation on the associated URL. The request method is passed to the Web Listener using the HTTP protocol.

2. The Web Listener starts the Web Agent.

   The Web Listener determines that the URL is associated with the Oracle Web Agent (by extracting the /owa portion of the URL), and spawns the Web Agent. The Web Listener then makes several environment variables available to the Web Agent in compliance with the CGI specification.

3. The Web Agent connects to the Oracle7 Server.

   In order to connect to an Oracle7 Server, the Web Agent requires certain information, such as which server to connect to and what username and password to use. This information is stored in the *owa.cfg file* (located in the OWSAdministrative directory) as part of a Web Agent service. The Web Agent determines which service to use by parsing the SCRIPT_NAME environment variable, which is set up by the Web Listener in accordance with the CGI 1.1 specification. The Web Listener derives the value of this environment variable from the path section just before the /owa in the URL. For example, if /ows-bin/hr/owa is the leading portion of the URL, then hr is the service to use.

   For information on how to set up a Web Agent service, please see the section , "Web Agent Service", later in this chapter.

4. The PL/SQL procedure is invoked.

   After connecting to the database, the Web Agent invokes the appropriate PL/SQL procedure, whose name is obtained by parsing the PATH_INFO environment variable.

   If the PL/SQL procedure requires parameters, these would have *been passed to the Web Agent by the Web Listener through the* QUERY_STRING environment variable or on standard input depending on the REQUEST METHOD ("GET" or "POST"). The Web Agent would then pass these on to the PL/SQL procedure.

   For more information on PATH_INFO, SCRIPT_NAME, and QUERY_STRING and how they are used by the Oracle Web Agent, please see the section, "How the Web Agent Uses CGI Environment Variables," later in this chapter.

MAC003146

ORCL00941822

5. The PL/SQL procedure executes, generating an HTML document.

   With the help of the Developer's Toolkit, the PL/ SQL procedure extracts data from the Oracle7 database and generates an HTML document in a PL/ SQL table.

   For more information on the Developer's Toolkit, please see Chapter 6, "Oracle WebServer Developer's Toolkit."

6. The Web Agent passes the HTML document produced to the Oracle Web Listener.

   If the PL/ SQL procedure terminates without an error, the Web Agent takes the generated HTML document from the PL/ SQL table and sends it to standard output, which is where the Web Listener expects CGI applications to send their output.

   If the PL/ SQL procedure returns with an error, the Web Agent sends a programmer–defined error page to standard output.

   For more information on how errors are handled, please see the section, "Oracle Web Agent Error Handling".

7. The Web Listener sends the HTML document to the requesting browser.

   The Web Listener returns the HTML document to the browser using HTTP. The browser is unaware that the document requested was generated on–the–fly and presents it on the screen like any other HTML document.

Web Agent Service

Since it is desirable that a single Web Listener be able to access multiple databases or schemas within a database, the Oracle Web Agent supports the concept of services. For more information on databases and schemas, see Chapter 4 of this manual. The concept of services is best described by an example:

Company A would like data stored in its Oracle7 database to be accessible to the public via an Oracle Web Listener. However, different departments within Company A have access to different parts of the database (schemas), which are partitioned by way of user names. Thus, the Human Resources department and the Accounts Payable department use different username/ password combinations to log on to the database. If both want to write CGI applications using the Oracle Web Agent to access their data, the Web Agent needs to use the correct username/ password when it logs on to the database. In this scenario, the owa.cfg file, which contains the configuration information, will

MAC003147

ORCL00941823

contain two services, an HR service, and an AP service. Each service will have an associated username/ password, an ORACLE_HOME parameter, and an ORACLE_SID parameter. When a request from a Web browser comes in, the Web Listener will extract the service name that is embedded in the URL and find out which parameters to use by reading the owa.cfg file.

To connect to the Oracle7 Server, the Web Agent needs the following information to be specified in the Web Agent service:

- username
- password
- ORACLE_HOME
- ORACLE_SID (for local databases only)
- SQL*Net V2 Service Name or Connect String (for remote databases only)

The Administration Utility allows the administrator to display, create, modify or delete a Web Agent service. With the Web Agent Creation form, you do not need to modify the configuration file for the Oracle Web Agent service (owa.cfg) directly. See the section "Creating or Modifying Web Agent Services," in Chapter 7 for more information.

The following is an example of a Web Agent service entry in the owa.cfg file, and a corresponding explanation for each parameter:

Developer's T#

```
(
owa_service = es
(
owa_user = www_es
)
(
owa_password = tiger
)
(
oracle_home = /opt/oracle7
)
(
oracle_sid = esprod
)
(
owa_err_page = /es_err.html
)
```


MAC003148

ORCL00941824

```
(
owa_valid_ports = 8000 8888
)
(
owa_log_dir = /opt/oracle7/ows/log
)
(
owa_nls_lang = AMERICAN_AMERICA.US7ASCII
)
)
```

owa_service

Name of the Web Agent service

owa_user

Database username that the Web Agent will use to connect to the database.

owa_password

Database password that the Web Agent will use to connect to the database.

oracle_home

The location of the Oracle7 code tree in the file system. This should be the ORACLE_HOME for the database to which this Web Agent service connects, unless the Web Agent service is set up to connect to a remote database (over SQL*Net). In that case, specify the ORACLE_HOME where the Web Agent is installed. The Web Agent is always installed in an ORACLE_HOME directory or some subdirectory thereof.

oracle_sid

Name of the database system ID to connect to. Does not apply for connections to a remote database.

owa_err_page

This is the path of the HTML document that is returned by the Web Agent when an error occurs in the PL/SQL procedure that the Web Agent invoked. This is the actual path, not the virtual path as configured in the Web Listener.

owa_valid_ports

The valid Web Listener network ports the Web Agent will service.

owa_log_dir

MAC003149

ORCL00941825

The directory where the Oracle Web Agent writes its error file. The error file will have the name service_name.err.

owa_nls_lang

The NLS_LANG of the Oracle7 database to which the Web Agent connects. If not specified, the Web Agent administration program looks up the database NLS_LANG when the the service is submitted.

How the Web Agent Uses CGI Environment Variables

The Web Agent uses the environment variables shown in Table 5 – 1:

Variable	Contains
REQUEST_METHOD	GET or POST
PATH_INFO	the name of PL/ SQL procedure to invoke
SCRIPT_NAME	contains the service the Web Agent is to use when logging on to Oracle7
QUERY_STRING	parameters to the PL/ SQL procedure (for GET method only. POST method parameters are passed via standard input.)

Table 5 – 1 CGI Variables Used by the Oracle Web Agent

The following is an example of how a typical URL is parsed by the Web Listener in compliance with the CGI 1.1 specification.

http://www.nhl.com:8080/ows-bin/nhl/owa/hockey.pass?person=Gretzky

1. The substring http://www.nhl.com:8080 in the above URL signals the Web browser to connect to the www.nhl.com host's port 8080 using the HTTP protocol.
2. When the Oracle Web Listener that is running on www.nhl.com receives the request, the substring /ows-bin/nhl/owa signals the Web Listener to execute the Web Agent instead of returning a file to the browser as it normally would have done with a static HTML document.
   Please see the section on CGI (Common Gateway Interface) in Chapter 3, "The Oracle Web Listener," for more information on how the Web Listener recognizes the difference between a CGI program and a static HTML document.
3. After spawning the Web Agent, the Web Listener passes /ows-bin/nhl/owa to the Web Agent as the environment variable

SCRIPT_NAME. The rest of the URL is passed to the Web Agent in this manner: The substring /hockey_pass gets passed to the Web Agent in PATH_INFO and the substring "person=Gretzky" is passed to the Web Agent in QUERY_STRING.

4. The Web Agent parses the SCRIPT_NAME to extract the service name nhl .
   - Note that if the Web Agent cannot resolve the passed service name, it defaults to owa_default_service.
5. The Web Agent parses PATH_INFO to determine that hockey_pass is the PL/ SQL procedure to invoke after logging on to Oracle7.
6. The Web Agent makes the parameters passed to it in QUERY_STRING available to the PL/ SQL procedure it invokes.

   Note: The fact that the QUERY_STRING environment variable is used indicates that this is a GET request. You can also pass parameters using the POST method. For an example of how to pass parameters using the POST method, please see the next section, "Passing Parameters to PL/ SQL".

Passing parameters to PL/SQL

A PL/ SQL procedure often requires parameters in order to execute and generate the appropriate HTML document. The following section discusses several key concepts and tips that a PL/ SQL developer should understand with respect to how parameters get passed to the specified PL/ SQL *routine*.

These key concepts and tips are:

- Getting Parameters from the Web Browser to the Web Agent
- Passing Parameters Using an HTML Form
- Ordering Parameters
- Defaulting Parameter Values
- Using Multiple HTML Form Fields with the Same Name
- Overloading Procedures

Getting Parameters from the Web Browser to the Web Agent

Depending on the REQUEST_METHOD used, parameters are passed from the Web Browser to the Web Listener to the Web Agent in one of two ways:

MAC003151

ORCL00941827

- QUERY_STRING environment variable — If the GET method is used by the Web browser, the Web Listener passes the parameters to the Web Agent in this environment variable, as shown in the example in the preceding section.
- standard input — If the POST method is used, the Web Listener passes the parameters to the Web Agent using standard input.

The method used to pass parameters from the Web Listener to the Web Agent is transparent to the PL/ SQL procedure that is the actual consumer of the parameter(s). This is an important feature of the Oracle Web Agent: the PL/ SQL programmer need not be aware of which method is used and need not be concerned with parsing either the QUERY_STRING environment variable or standard input. Thus, the PL/ SQL programmer can concentrate on what he or she knows best: developing the logic to extract data from the Oracle database, based on pre–parsed parameters passed by the Oracle Web Agent.

It is recommended that you use POST whenever possible. GET is the method used for links and non–form URLs. For HTML forms, one has a choice. Because the GET method uses operating system environment variables, there are limits on the length of the QUERY_STRING.

Passing Parameters Using an HTML Form

The following example is analogous to the one in the previous section, except that it uses an HTML form that employs the POST REQUEST_METHOD.

```
<FORM METHOD="POST"
ACTION="http://www.nhl.com:8080/ows-bin/nhl/owa/hockey.pass">
Please type the name of the person you wish to search for:
<INPUT TYPE="text" NAME="person"><P>
To submit the query, press this button:
<INPUT TYPE="submit" VALUE="Submit Query">. <P>
</FORM>
```

The above form will cause the Oracle Web Listener to behave in the same fashion as the previous example, except that instead of populating the QUERY_STRING environment variable with "person=Gretzky," the Web Listener will write "person=Gretzky" to standard input. This assumes, of course, user typed "Gretzky" in the entry field of the HTML form above.

Note that the name of the HTML input variable, in this case "person", has to be the same as the PL/ SQL parameter it is to match.

The PL/ SQL procedure that is the recipient of the above parameters follows:

MAC003152

ORCL00941828

```
create or replace
procedure hockey_pass (person in varchar2) is
   n_assists integer;
begin
   select num_assists into n_assists
      from hockey_stats
   where name=person;
   htp.print(person||' has '||to_char(n_assists)||' assists this
   season');
end;
```

Ordering Parameters

Generally, the PL/ SQL developer does not need to be concerned with the order in which the Oracle Web Agent receives parameters from an HTML form or through a URL. The only case that it will be relevant is when passing multiple values for the same form field. See "Using Multiple HTML Form Fields with the Same Name" later in this section.

Defaulting Parameter Values

If the PL/ SQL developer cannot guarantee that a value will be passed from a Web Browser for a particular PL/ SQL procedure parameter, then it is recommended that he or she give the parameter a default value. For example:

```
create or replace procedure showvals(a in varchar2 DEFAULT NULL,
                                     b in varchar2 DEFAULT NULL)
is
begin
   htp.print('a = '||a||htp.br);
   htp.print('b = '||b||htp.br);
end;
```

If the Web Agent were to receive a request to call procedure showvals where there was no value for "a", the value for "b" was, say, "Hello", and the DEFAULT NULL clause was not part of the procedure's definition, then the request would generate an error with the following message:

```
OWS-05111: Agent : no procedure matches this call
  OWA SERVICE: test_service
  PROCEDURE: showvals
 PARAMETERS:
  ===========
  B:
   Hello
```


MAC003153

ORCL00941829

By "defaulting" the parameters, the above request would properly output:

a =

b = Hello

which to the end user would look like:

a =
b = Hello

Using Multiple HTML Form Fields with the Same Name

There are a number of instances where one would want to have multiple values passed for the same HTML form variable, and hence to the same PL/SQL parameter. To handle this situation, one can use PL/SQL tables to create an array of values.

One case where multiple values are passed with the same HTML form variable name is in the use of the HTML form tag "SELECT". If one sets the SIZE parameter to something greater than 1, then the user will be able to select multiple values for the same form field.

Another case where one has a set of values corresponding to a single form field follows:

```
-- QUERY_FORM prints an HTML page with all the columns for the
-- specified table.  Invoke the procedure from a Web Browser with
-- a URL like:
http://yourhost:port_num/service_name/owa/query_form?the_table=emp

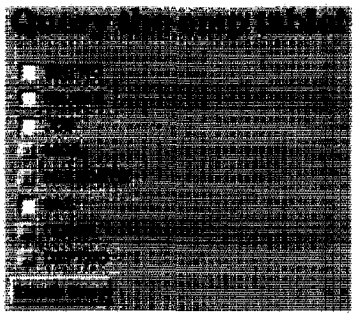
create or replace procedure query_form(the_table in varchar2) is
    cursor cols is
    select column_name
        from user_tab_columns
    where table_name = upper(the_table);
begin
    htp.htmlOpen;
    htp.headOpen;
    htp.htitle('Query the '||the_table||' table!');
    htp.headClose;
    htp.bodyOpen;
    -- Use owa_util.get_owa_service path to automatically retrieve
    htp.formOpen(owa_util.get_owa_service_path||'do_query');
    -- Put in the table as a hidden field to pass on to do_query
    htp.formHidden('the_table', the_table);
    -- Put in a dummy value, as we cannot DEFAULT NULL a PL/SQL
```


MAC003154

ORCL00941830

```
    table.
htp.formHidden('COLS', 'dummy');
    for crec in cols loop
        -- Create a checkbox for each column. The form field name
        -- will be COLS and the value will be the given column name.
        -- Will need to use a PL/SQL table to retrieve a set of
        -- values like this. Can use the owa_util.ident_arr type
        -- since the columns are identifiers.
        htp.formCheckbox('COLS', crec.column_name);
        htp.print(crec.column_name);
        htp.nl;
    end loop;
    -- Pass a NULL field name for the Submit field; that way, a
    -- name/value pair is not sent in. Wouldn't want to do this
    -- if there were multiple submit buttons.
    htp.formSubmit(NULL, 'Execute Query');
    htp.formClose;
    htp.bodyClose;
    htp.htmlClose;
end;
```

Invoking this procedure will bring up a page which looks like:



In this example, the user has already selected to query the EMPNO, ENAME, JOB, and SAL columns:

Here is a procedure to process this form submission:

```
-- DO_QUERY executes the query on the specified columns and
-- tables. The OWA_UTIL.IDENT_ARR datatype is defined as:
--   -- type ident_arr is table of varchar2(30) index by
binary_integer
-- create or replace procedure do_query(the_table in varchar2,
                                        cols in owa_util.ident_arr) is

    column_list  varchar2(32000);
    col_counter  integer;
```

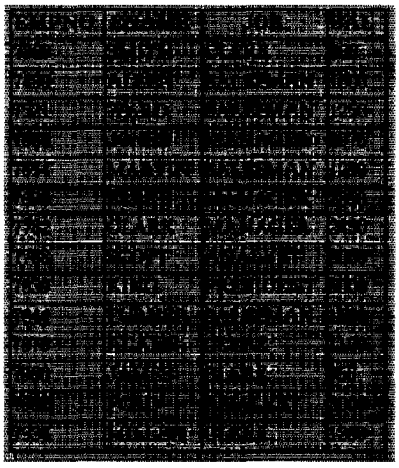

MAC003155

ORCL00941831

```
   ignore        boolean;
begin
-- For PL/SQL tables, have to just loop through until you hit
-- no_data_found. Start the counter at 2 since we put in
-- a dummy hidden field.
   col_counter := 2;
   loop
      -- build a comma-delimited list of columns
      column_list := column_list||cols(col_counter)||',';
      col_counter := col_counter + 1;
   end loop;
exception
   when no_data_found
   then
       -- strip out the last trailing comma
       column_list := substr(column_list,1,length(column_list)-1);

        -- print the table - assumes HTML table support
       ignore := owa_util.tablePrint(the_table, 'BORDER',
                                              OWA_UTIL.HTML_TABLE,
 column_list);
end;
```

Then after selecting the "Execute Query" button, the user would see:



It is a good idea to use a hidden place-holder variable as the first value if you cannot guarantee that at least one value will be submitted for the PL/SQL table. The reason is that one cannot DEFAULT a PL/SQL table. And a call to this procedure with just one argument (the_table) would cause the Web Agent to generate an error.

MAC003156

ORCL00941832

Note that the Web Agent can only pass parameters to PL/ SQL tables which have a base type of VARCHAR2. This should not provide a significant limitation, as the PL/ SQL type VARCHAR2 is the largest PL/ SQL datatype with a maximum length of 32767 bytes. The values can then be explicitly converted to NUMBER, DATE, or LONG within a stored procedure (using TO_NUMBER or TO_DATE – no conversion needed for LONGs).

Overloading Procedures

PL/ SQL allows developers to overload procedures and functions that are in PL/ SQL packages (but not standalone functions and procedures, For a discussion of overloading as implemented in PL/ SQL, see "Overloading Subprograms" in Chapter 4 of this manual). For example:

```
create or replace package overload is
      procedure proc1(charval  in varchar2);
      procedure proc1(numval   in number);
end;

create or replace package body overload is
      procedure proc1(charval  in varchar2) is
      begin
         htp.print('The character value is '||charval);
      end;

      procedure proc1(numval   in number);
         htp.print('The number value is '||numval);
      end;
end;
```

This functionality can be utilized by the Web Agent, but with the restriction that procedures that are overloaded on datatypes should not have the same parameter names. For example:

```
create or replace package overload is
      procedure proc1(val  in varchar2);
      procedure proc1(val  in number);
end;
```

When the Web Agent attempts to determine which procedure to call, it will not be able to distinguish between the two and will generate an error.

This limitation is imposed by the lack of HTML–form datatypes, but should not provide a significant limitation, as the PL/ SQL type VARCHAR2 is the largest PL/ SQL datatype with a maximum length of 32767 bytes. The values can then be explicitly converted to NUMBER,

MAC003157

ORCL00941833

DATE, or LONG within a stored procedure (using TO_NUMBER or TO_DATE – no conversion needed for LONGs).

Accessing CGI Environment Variables

The Oracle Web Listener conforms to the Common Gateway Interface (CGI) 1.1 specification. Thus, all CGI environment variables that are part of this specification are passed from the Oracle Web Listener to the Web Agent. The Web Agent, in turn, makes all relevant environment variables accessible from within PL/ SQL.

The Web Agent makes available the CGI environment variables shown in Table 5 – 2.

Variable	Variable Meaning
AUTH_TYPE	Method used to validate user
GATEWAY_INTERFACE	The revision of the CGI specification to which the server complies
HTTP_USER_AGENT	The browser the client is using to send the request
PATH_INFO	Extra path information given by the client
PATH_TRANSLATED	Translated version of PATH_INFO provided by server for mapping. This consists of the document root concatenated to the PATH_INFO.
REMOTE_HOST	Hostname making the request if it can be determined
REMOTE_ADDR	IP address of the remote host making the request
REMOTE_USER	Used to authenticate user
REMOTE_IDENT	Set to the remote username retrieved from the server
SERVER_PROTOCOL	Name and revision of the information protocol used in the request
SERVER_SOFTWARE	Name and version of information server software answering the request
SERVER_NAME	The server's hostname, or IP address

MAC003158

ORCL00941834

SERVER_PORT	Port number on which the server is running
SCRIPT_NAME	Virtual path to the script being executed, used for self–referencing URL

Table 5 – 2 CGI Environment Variables

These environment variables can be accessed from within PL/ SQL using the owa_util.get_cgi_env function, which is documented in Chapter 6," The Oracle WebServer Developer's Toolkit".

Oracle Web Agent Error Handling

There are two types of errors that are handled by the Oracle Web Agent:

- application errors
- system errors

Application Errors

Application errors are specific to the PL/ SQL application. All applications written by a programmer should have their own exception handling in the PL/ SQL procedure that produces the appropriate output.

Because the Oracle Web Agent does not read the HTML output to determine its content, handled exceptions are transparent. As far as the Web Agent is concerned, if the PL/ SQL code generates HTML output, the operation was successful. The user will see whatever handled exception message is generated by the PL/ SQL procedure.

System Errors

System errors are detected by the Oracle Web Agent itself. These are errors that occur when the Web Agent is unable to launch the PL/ SQL procedure or when a PL/ SQL exception is not handled by the stored procedure, causing the exception to be propagated back to the Web Agent as a system error. This causes a standard HTML error document to be returned to the browser.

For example, if the Oracle Web Agent cannot make a connection to the Oracle7 Server, the PL/ SQL procedure cannot run and a system error occurs. The Web Agent then returns a default error message to the browser from the HTTP Server, or returns a customized HTML error

MAC003159

ORCL00941835

page (if one was previously configured as part of the Web Agent service using the OWA_ERR_PAGE parameter).

How the Web Agent Knows What Error Page to Generate

The Web Agent checks for the error page field in the OWA service configuration. If a value is found there, the Web Agent looks for the specified page in the file system. If found, that page is returned to the Web client. If the lookup fails, a default error message is returned to the Web client.

Creating Error Pages

The Oracle WebServer allows the administrator to create customized error pages that are returned whenever the Web Agent encounters a system error. The purpose of customized error pages is to tell the user what action to take.

Only one custom error page per service can be returned when the Web Agent encounters a system error.

To create a custom HTML error page, use any text or HTML editor you are comfortable with to write your HTML document.

The following is an example of a user–defined HTML error page for a system error:

```
<HTML>
<HEAD>
<TITLE>ERROR</TITLE>
</HEAD>
<BODY>
<HR>
<H1>Your Request Failed</H1>
<HR>
Bob's Big Bulldozer Company could not process your request. Please
try again later, or call us at 1–800–TRACTOR
Thanks, Bob
</BODY>
</HTML>
```

Once the customized HTML error page is created, the entry for OWA error page in the Web Agent Service configuration file must reflect where the new HTML error page is stored. Use the Web Service Administration page to insert or change the OWA error page entry.

A full path must be specified for the customized HTML error page. If a path is not specified, the Web Agent will search for it in the same directory in which the Web Agent executable is stored.

MAC003160

ORCL00941836

See the section, "Creating or Modifying the Web Agent Service," in Chapter 7, "Oracle WebServer Administration Utility," for more information.

Error Log File

The Oracle Web Agent keeps an error log file, defined by the OWA Service Name and the OWA Log Directory value for that service. The file will be named service_name.err. An entry is appended to this file whenever the Oracle Web Agent encounters an error. The error log records all relevant information, such as date, error number, and URL.

Remember, there is only one error log for each Web Agent service. However, if the Web Agent cannot resolve a service name, and there is no OWA_DEFAULT_SERVICE configured, then errors will be written to OWA.err in the default log directory.

The following is an example of an error log file, and the corresponding meaning for each line in the file:

```
Wed Jun 28 08:14:27 1995                          /*Timestamp*/
OWS-05100: Agent : unable to connect due to Oracle error 1017
ORA-01017: invalid username/password; logon denied /*Error stack*/
OWA SERVICE: HR                                   /*Web Agent Service*/
PROCEDURE: hrinfo.emp                             /*PL/SQL procedure called*/
PARAMETERS:                                       /*parameters used by procedure*/
============
ENAME:
RayBorque
```

Oracle Web Agent Version Number

To find out what version of the Oracle Web Agent is installed on your operating system use the following procedure.

At the command line enter the following: (UNIX only)

$ORACLE_HOME/ows/bin/owa -v

The command will return the following (your result may vary):

```
Oracle Web Agent Release 1.0.0.0.0
```


MAC003161

ORCL00941837

MAC003162

ORCL00941838

CHAPTER

6

Oracle WebServer Developer's Toolkit

This chapter describes the hypertext procedures, hypertext functions, and utilities that make up the Oracle WebServer Developer's Toolkit.

One of the main goals of the Oracle Web Agent is to eliminate the PL/ SQL programmer's need to be intimately familiar with World Wide Web technology. To this end, the Oracle WebServer includes a Developer's Toolkit made up of several PL/ SQL packages that minimize the programmer's need to know HTML syntax. Although the programmer is still required to have a working knowledge of HTML, by using the Toolkit he or she will not need to hard code the exact syntax of HTML tags into PL/ SQL procedures. For instance, a programmer still needs to realize that an anchor tag is needed, but he or she doesn't need to know the exact sequence of characters needed to generate an anchor.

The Oracle WebServer Developer's Toolkit includes the following PL/ SQL Packages:

- Hypertext Procedures (HTP)

  A hypertext procedure generates a line in an HTML document that contains the HTML tag that corresponds to its name. For instance, the htp.anchor procedure generates an anchor tag. The HTP package will be the most commonly used package of the three.

MAC003163

ORCL00941839

- Hypertext Functions (HTF)

  A hypertext function returns the HTML tag that corresponds to its name. However, it is not sufficient to call an HTF function on its own because the HTML tag is not passed to the Web Agent. The output of an HTF function must be passed to htp.print in order to actually be part of an HTML document. Thus, the following line:

  ```
  htp.print(htf.italic('Title'));
  ```

  is functionally equivalent to:

  ```
  htp.italic('Title');
  ```

  Every hypertext function (HTF) has a corresponding hypertext procedure (HTP). Thus, HTF functions are generally used only when the programmer needs to nest calls, such as:

  ```
  htp.header(1, htf.italic('Title'));
  ```

  In this example, 'htf.italic' will generate the following character string:

  ```
  <I>Title</I>
  ```

  This string is then passed to the 'htp.header' procedure and the following line will appear in the HTML document being formatted:

  ```
  <H1><I>Title</I></H1>
  ```

- OWA Utilities (OWA_UTIL)

  A collection of useful utility procedures and functions. The purposes of these range from printing a signature tag on HTML pages to easy formatting of Oracle tables into HTML.

- OWA

  A set of procedures called only by the Oracle Web Agent itself. None of the subprograms in this package should be called directly from user–developed PL/SQL.

Installing the Oracle WebServer Developer's Toolkit

To install the Developer's Toolkit, use the Oracle Web Agent Service Administration forms. Both the Service Creation and Service Modification pages provide a checkbox for installing the Developer's Toolkit. Selecting this button and submitting the form will do the following:

MAC003164

ORCL00941840

- grant the CONNECT and RESOURCE roles to the OWA database user For more information on these roles or on granting roles, see "GRANT (roles)" in Chapter 4 of the Oracle7 Server SQL Reference.
- execute the OWAINS.SQL sql script, which can be found in the ows Administration directory. If run manually, the script should be run from SQL*DBA or from Server Manager. If you want to run it from SQL*Plus, see the header of the script for instructions.

  The OWAINS.SQL script installs all of the Developer's Toolkit packages.

Optimizing Multiple–Service Installations

If your site has multiple Web Agent services, you can minimize the amount of storage space used and enhance PL/ SQL performance by doing the following:

1. Install the Developer's Toolkit in one database user's schema. This user becomes the toolkit owner.
2. Drop the Developer's Toolkit PL/ SQL from the schemas of the OWA database users for other Web Agent services, if you have already installed them.

```
connect <user> / <password>
drop package HTF;
drop package HTP;
drop package OWA_UTIL;
drop package OWA;
```

3. Grant the system privilege EXECUTE on the PL/ SQL packages to OWA database users for other Web Agent services. For more information on this command, see "GRANT (system privileges)" in Chapter 4 of the Oracle7 Server SQL Reference.

```
connect <toolkit owner> / <password>
grant execute on HTF to <user>;
grant execute on HTP to <user>;
grant execute on OWA_UTIL to <user>;
grant execute on OWA to <user>;
```

4. Create synonyms for the Developer's Toolkit PL/ SQL packages in the schemas of all OWA database users who are specified in Web Agent services. Synonyms are alternate names that make it possible for the packages to be referred to without being qualified by schema names. For more information, see "Ownership and Naming Conventions" in Chapter 4 of this manual and see "CREATE SYNONYM" in Chapter 4 of the Oracle7 Server SQL Reference.

MAC003165

ORCL00941841

```
connect <user>/<password>
create synonym HTF for <Toolkit owner>.HTF;
create synonym HTP for <Toolkit owner>.HTP;
create synonym OWA_UTIL for <Toolkit owner>.OWA_UTIL;
create synonym OWA for <Toolkit owner>.OWA;
```

Security Note

PL/SQL procedures run with the privileges of the creator of the PL/SQL code. For the Developer's Toolkit, this is only an issue for the owa_util package. Two of the subprograms, showsource and tableprint, access user data. Granting execute privileges on this package to users allows those users to view the tables, views, and stored PL/SQL code of the owner of owa_util.

If this is a security issue for your installation, install the owa_util package separately for each OWA database user.

The scripts to do this are:

- PUBUTIL.SQL
- PRIVUTIL.SQL

Both scripts reside in the OWSAdministration directory, and should be run in the order listed.

Procedure and Function Reference

This section describes each procedure and function in the htp, htf, and owa_util packages. Please note that for every htp procedure that generates HTML tags, a corresponding htf function exists with identical parameters. Note that defaulted parameters do not need to be passed.

The description of each procedure or function is broken down into the following parts:

Name	Name of procedure or function
Syntax	Actual syntax of procedure or function
Purpose	What the procedure or function does
Parameters	Parameters passed in
Generates	What the procedure or function generates

MAC003166

ORCL00941842

Parameters Passed into Procedures and Functions

All parameters passed into a hypertext procedure or function are of data type varchar2 (varying–length character string), integer, or date. The data type is indicated by a "c" for character, (varchar2) "n" for integer, or "d" for date as the first character of the parameter name. For example:

- cname in varchar2

  where cname is of data type character

- nsize in integer

  where nsize is of data type integer

- dbuf in date

  where dbuf is a date field

Note: Many HTML 3.0 tags have a large number of optional attributes that, if made as individual parameters to the hypertext procedures or functions would make the calls quite cumbersome. In addition, some browsers support non–standard attributes. Therefore, each hypertext procedure or function that generates an HTML tag has as its last parameter cattributes, an optional parameter. This parameter enables you to pass the exact text of the desired HTML attributes to the PL/SQL procedure.

For example, the syntax for htp.em is:

```
htp.em(ctext, cattributes);
```

A call that uses HTML 3.0 attributes might look like the following:

```
htp.em('This is an example','ID="SGML_ID" LANG="en"');
```

This line would generate the following:

```
<EM ID="SGML_ID" LANG="en">This is an example</EM>
```

Print Procedures

The following print procedures are used in conjunction with htf functions to generate a line in the HTML document being constructed. They can also be passed hard–coded text that will appear in the HTML document as–is. The generated line is passed to the Web Agent which sends it to standard output. As documented in the CGI 1.1 specification, the Oracle Web Listener takes the contents of standard output and returns it to the Web browser that requested the dynamic HTML document.

MAC003167

ORCL00941843

htp.print

Syntax	htp.print (cbuf | dbuf | nbuf);
Purpose	generates a line in an HTML document.
Parameters	cbuf in varchar2 or dbuf in date or nbuf in number
Generates	Generates a line in an HTML document based on the value passed to it.

htp.p Alias for htp.print

htp.prn

Syntax	htp.prn (cbuf | dbuf | nbuf);
Purpose	Just like htp.print, but doesn't put a new line at the end of the value submitted.

htp.prints

Syntax	htp.prints (ctext);
Purpose	Generates a line in an HTML document and replaces all occurrences of the following special characters with escape characters. If not replaced, the special characters would be interpreted as HTML control characters, and would produce garbled output. '<' with '<' '>' with >' '"' with '"' '&' with '&'
Parameters	ctext in varchar2
Generates	Generates a line in an HTML document based on the value passed to it. This procedure is the same as htp.print or htp.p but first replaces the special characters listed above with escape characters.

htp.ps Alias for htp.prints

MAC003168

ORCL00941844

Structure Tags

The following tags are used to identify the major parts of an HTML document.

> *Note: Although this section shows hypertext procedures (HTP), all of them are also available as hypertext functions (HTF).*

htp.htmlOpen

Syntax	htp.htmlOpen;
Purpose	Prints a tag that indicates the beginning of an HTML document
Parameters	none
Generates	<HTML>

htp.htmlClose

Syntax	htp.htmlClose;
Purpose	Prints a tag that indicates the end of an HTML document
Parameters	none
Generates	</ HTML>

htp.headOpen

Syntax	htp.headOpen;
Purpose	Prints a tag that indicates the beginning of the HTML document head
Parameters	none
Generates	<HEAD>

htp.headClose

Syntax	htp.headClose;
Purpose	Prints a tag that indicates the end of the HTML document head
Parameters	none

MAC003169

ORCL00941845

Generates	</HEAD>

htp.bodyOpen

Syntax	htp.bodyOpen (cbackground, cattributes);
Purpose	Prints the tag that identifies the beginning of the body of an HTML document, and allows you to specify an image as the background of the document
Parameters	cbackground in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<BODY background="*cbackground*" *cattributes*> Note: If cbackground and cattributes are NULL, this tag generates <BODY>.
Example	htp.bodyOpen ('/img/background.gif'); This line produces: <BODY background="background.gif">

htp.bodyClose

Syntax	htp.bodyClose;
Purpose	Defines the end of the HTML document body
Parameters	none
Generates	</BODY>

Head Related Tags

The following procedure tags should be placed between the htp.headOpen and htp.headClose procedure tags.

> Note: Although this section shows hypertext procedures (HTP), all of them are also available as hypertext functions (HTF).

htp.title

Syntax	htp.title (ctitle);
Purpose	Prints an HTML tag with the text you pass in as the value of TITLE. Most Web Browsers display the text value enclosed between <TITLE> and </TITLE> at the top of the document viewing window.
Parameters	ctitle in varchar2

MAC003170

ORCL00941846

Generates	<TITLE>*ctitle*</TITLE>

htp.htitle

Syntax	htp.htitle (ctitle, nsize, calign, cnowrap, cclear, cattributes);
Purpose	Prints the HTML tags for both the title and a top heading using the title you specify.

Parameters

ctitle	in varchar2	
nsize	in integer	DEFAULT 1
calign	in varchar2	DEFAULT NULL
cnowrap	in varchar2	DEFAULT NULL
cclear	in varchar2	DEFAULT NULL
cattributes	in varchar2	DEFAULT NULL

Generates

<TITLE>*ctitle*</TITLE><H*nsize* ALIGN="calign" NOWRAP CLEAR="*cclear*" *cattributes*>ctitle</H*nsize*>

Note that htp.htitle produces non–standard HTML. However, most Web Browsers accept it. To produce standard HTML, the <TITLE> tag must be in the HEAD section of the page, and the level heading tag (<Hn>) must be in the BODY section. For example:

```
htp.headOpen;
htp.title(...);
htp.headClose;

htp.bodyOpen;
htp.header (1,...));
...
```

htp.base

Syntax	*htp.base;*
Purpose	Prints an HTML tag that records the URL of the document
Parameters	none
Generates	Inserts absolute pathname of current document.

MAC003171

ORCL00941847

htp.isindex

Syntax	htp.isindex (cprompt, curl);		
Purpose	Creates a single entry field with a prompting text, such as "enter value," then sends that value to the URL of the page or program.		
Parameters	cprompt	in varchar2	DEFAULT NULL
	curl	in varchar2	DEFAULT NULL
Generates	<ISINDEX PROMPT="*cprompt*" HREF="*curl*">		

htp.linkRel

Syntax	htp.linkRel (crel, curl, ctitle);		
Purpose	Prints the HTML tag that gives the relationship described by the hypertext link from the anchor to the target. This is only used when the HREF attribute is present. This tag indicates a relationship between documents, but does not create a link. To do that, use htp.anchor.		
Parameters	crel	in varchar2	
	curl	in varchar2	
	ctitle	in varchar2	DEFAULT NULL
Generates	<LINK REL="*crel*" HREF="*curl*" TITLE="*ctitle*">		

htp.linkRev

Syntax	htp.linkRev (crev, curl, ctitle);		
Purpose	Gives the relationship described by the hypertext link from the target to the anchor. This is the opposite of htp.linkRel. This tag indicates a relationship between documents, but does not create a link. To do that, use htp.anchor.		
Parameters	crev	in varchar2	
	curl	in varchar2	
	ctitle	in varchar	DEFAULT NULL
Generates	<LINK REV="*crev*" HREF="*curl*" TITLE="*ctitle*">		

htp.meta

Syntax	htp.meta (chttp_equiv, cname, ccontent);
Purpose	

MAC003172

ORCL00941848

Prints an HTML tag that identifies and embeds document meta–information that supplies the Web browser with information about the objects returned in HTTP.

Parameters		
	chttp_equiv	in varchar2
	cname	in varchar2
	ccontent	in varchar2

Generates

```
<META HTTP-EQUIV="chttp_equiv" NAME="cname"
  CONTENT="ccontent">
```

Example

```
htp.meta ('Refresh', NULL, 120);
```

This line produces:

```
<META HTTP-EQUIV="Refresh" CONTENT=120>
```

which on some Web browsers will cause the current URL to be reloaded automatically every 120 seconds.

Body Tags

Body tags are used in the main text of your HTML page. They can format a paragraph, allow you to add hidden comments to your text, and add images within the body of your HTML text.

> Note: Although this section shows hypertext procedures (HTP), all of them are also available as hypertext functions (HTF).

htp.line

Syntax: htp.line (cclear, csrc, cattributes);

Purpose: Prints the HTML tag that generates a line in the HTML document. csrc enables you to specify a custom image as the source of the line.

Parameters			
	cclear	in varchar2	DEFAULT NULL
	csrc	in varchar2	DEFAULT NULL
	cattributes	in varchar2	DEFAULT NULL

Generates

```
<HR CLEAR="cclear" CSRC="csrc" cattributes>
```

htp.hr

Alias for htp.line

htp.nl

Syntax: htp.nl (cclear, cattributes);

MAC003173

ORCL00941849

Purpose	Prints the HTML tag that inserts a new line		
Parameters	cclear	in varchar2	DEFAULT NULL,
	cattributes	in varchar2	DEFAULT NULL,
Generates	<BR CLEAR="*cclear*" *cattributes*>		

htp.br Alias for htp.nl

htp.header

Syntax	htp.header (nsize, cheader, calign, cclear, cattributes);		
Purpose	Prints the HTML tag for a heading level, with the value of the heading level assigned in the nsize parameter. Valid levels are 1 through 6.		
Parameters	nsize	in integer	
	cheader	in varchar2	
	calign	in varchar2	DEFAULT NULL
	cnowrap	in varchar2	DEFAULT NULL
	cclear	in varchar2	DEFAULT NULL
	cattributes	in varchar2	DEFAULT NULL
Generates	<H*nsize* ALIGN="*calign*" NOWRAP CLEAR="*cclear*" *cattributes*>*cheader*</H*nsize*>		
Example	htp.header (1, 'Overview'); produces <H1>Overview</H1>		

htp.anchor

Syntax	htp.anchor (curl, ctext, cname, cattributes);		
Purpose	Prints the HTML tag for an anchor to be the start or end destination of a hypertext link. This anchor can accept several attributes, but either HREF or NAME is required. HREF specifies where to link to. NAME allows this tag to be a target of a hypertext link.		
Parameters	curl	in varchar2	
	ctext	in varchar2	
	cname	in varchar2	DEFAULT NULL
	cattributes	in varchar2	DEFAULT NULL
Generates	<A HREF="*curl*" NAME="*cname*" *cattributes*>*ctext*</A>		

htp.mailto

Syntax	htp.mailto (caddress, ctext, cname, cattributes);
Purpose	Prints the HTML tag for an anchor with 'mailto' concatenated ahead of the mail address argument.

Parameters			
	caddresss	in varchar2	
	ctext	in varchar2	DEFAULT NULL
	cattributes	in varchar2	DEFAULT NULL

Generates	<A HREF="mailto:*caddress*" *cattributes*>*ctext*</A>
Example	htp.mailto('pres@white_house.gov','Send Email to the President'); prints <A HREF="mailto:pres@white_house.gov">Send Email to the President</A>

htp.img

Syntax	htp.img (curl, calign, calt, cismap, cattributes);
Purpose	Prints an HTML tag that signals the browser to load an image to be placed into the HTML page. ALT allows you to specify alternate text to be shown while the image is being loaded, or instead of the image if the browser does not support images. The ISMAP attribute indicates that the image is an image map.

Parameters			
	curl	in varchar2	DEFAULT NULL
	calign	in varchar2	DEFAULT NULL
	calt	in varchar2	DEFAULT NULL
	cismap	in varchar2	DEFAULT NULL
	cattributes	in varchar2	DEFAULT NULL

Generates	<IMG SRC="*curl*" ALIGN="*calign*" ALT="*calt*" ISMAP *cattributes*>

htp.para

Syntax	htp.para;
Purpose	Prints an HTML tag that indicates that the text previous to it should be formatted as a paragraph.
Parameters	none
Generates	<P>

MAC003175

ORCL00941851

htp.paragraph

Syntax	htp.paragraph (calign, cnowrap, cclear, cattributes);
Purpose	Prints the same HTML tag as htp.para except that parameters pass in exact alignment, leading, wrapping , and attributes.
Parameters	calign in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<P ALIGN="*calign*" NOWRAP CLEAR="*cclear*" *cattributes*>

htp.address

Syntax	htp.address (cvalue, cnowrap, cclear, cattributes);
Purpose	Prints an HTML tag that enables you to specify address, author and signature of document
Parameters	cvalue in varchar2 cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<ADDRESS CLEAR="*cclear*" NOWRAP *cattributes*>*cvalue*</ADDRESS>

htp.comment

Syntax	htp.comment (ctext);
Purpose	Prints an HTML tag that allows you to store comments or lines in HTML pages. These comments are not visible to the end user.
Parameters	ctext in varchar2
Generates	<!— *ctext* —>

htp.preOpen

Syntax	htp.preOpen (cclear, cwidth, cattributes);
Purpose	Prints an HTML tag that indicates the beginning of preformatted text in the body of the HTML page.

MAC003176

ORCL00941852

Parameters	cclear	in varchar2 DEFAULT NULL
	cwidth	in varchar2 DEFAULT NULL
	cattributes	in varchar2 DEFAULT NULL

<PRE CLEAR="*cclear*" WIDTH="*cwidth*" *cattributes*>

htp.preClose

Syntax	htp.preClose;
Purpose	Prints an HTML tag that ends the preformatted section of text.
Parameters	none
Generates	</PRE>

htp.blockquoteOpen

Syntax	htp.blockquoteOpen (cnowrap, cclear, cattributes);	
Purpose	Prints an HTML tag that precedes a paragraph of quoted text.	
Parameters	cnowrap	in varchar2 DEFAULT NULL
	cclear	in varchar2 DEFAULT NULL
	cattributes	in varchar2 DEFAULT NULL;
Generates	<BLOCKQUOTE CLEAR="*cclear*" NOWRAP *cattributes*>	

htp.blockquoteClose

Syntax	htp.blockquoteClose;
Purpose	Ends the <BLOCKQUOTE> section of quoted text.
Parameters	none
Generates	</BLOCKQUOTE>

List Tags

List tags allow you to display information in any of the following ways:

- ordered: these lists have numbered items
- unordered: these lists have bullets to mark each item
- definition: these lists alternate a term with its definition

MAC003177

ORCL00941853

Note: All the hypertext procedures (HTP) shown in this section are also available as hypertext functions (HTF).

htp.listHeader

Syntax	htp.listHeader (ctext, cattributes);
Purpose	Prints an HTML tag at the beginning of the list
Parameters	ctext in varchar2 cattributes in varchar2 DEFAULT NULL
Generates	<LH *cattributes*>*ctext*</LH>

htp.listItem

Syntax	htp.listItem (ctext, cclear, cdingbat, csrc, cattributes);
Purpose	Prints an HTML tag that formats a listed item.
Parameters	ctext in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cdingbat in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<LI CLEAR="*cclear*" DINGBAT="*cdingbat*" SRC="*csrc*" *cattributes*>*ctext*

htp.ulistOpen

Syntax	htp.ulistOpen (cclear, cwrap, cdingbat, csrc, cattributes);
Purpose	Prints an HTML tag that is used to open an unordered list that presents listed items separated by white space and marked off by bullets.
Parameters	cclear in varchar2 DEFAULT NULL cwrap in varchar2 DEFAULT NULL cdingbat in varchar2 DEFAULT NULL csrc in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<UL CLEAR="*cclear*" WRAP="*cwrap*" DINGBAT="*cdingbat*" SRC="*csrc*" *cattributes*>

MAC003178

ORCL00941854

htp.ulistClose

Syntax	htp.ulistClose;
Purpose	Prints an HTML tag that ends the unordered list.
Parameters	none
Generates	</UL>

htp.olistOpen

Syntax	htp.olistOpen (cclear, cwrap, cattributes);
Purpose	Prints an HTML tag that is used to open an ordered list that presents listed items marked off with numbers.
Parameters	cclear in varchar2 DEFAULT NULL cwrap in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<OL CLEAR="*cclear*" WRAP="*cwrap*" *cattributes*>

htp.olistClose

Syntax	htp.olistClose;
Purpose	Prints an HTML tag that ends an ordered list.
Parameters	none
Generates	</OL>

htp.dlistOpen

Syntax	htp.dlistOpen (cclear, cattributes);
Purpose	Prints an HTML tag that starts a definition list
Parameters	cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<DL CLEAR="*cclear*" *cattributes*>

htp.dlistClose

Syntax	htp.dlistClose
Purpose	Prints an HTML tag that Ends a definition list
Parameters	none

MAC003179

ORCL00941855

Generates	</DL>

htp.dlistDef

Syntax	htp.dlistDef (ctext, cclear, cattributes);
Purpose	Prints an HTML tag that is used to insert terms, and their corresponding definitions in an indented list format. The htp.dlistTerm must immediately follow this tag.
Parameters	ctext in varchar2 DEFAULT NULL clear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<DD CLEAR="*cclear*" *cattributes*>*ctext*

htp.dlistTerm

Syntax	htp.dlistTerm (ctext, cclear, cattributes);
Purpose	Prints an HTML tag used to insert the definition term inside the definition list. This tag must immediately follow the htp.dlistDef.
Parameters	ctext in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<DT CLEAR="*cclear*" *cattributes*>*ctext*

htp.menulistOpen

Syntax	htp.menulistOpen;
Purpose	Prints an HTML tag that begins a list that presents one line per item, and appears more compact than an unordered list. The htp.listItem will follow this tag.
Parameters	none
Generates	<MENU>

htp.menulistClose

Syntax	htp.menulistClose;
Purpose	Prints an HTML tag that ends a menu list.

MAC003180

ORCL00941856

Paramenters	none
Generates	</MENU>

htp.dirlistOpen

Syntax	htp.dirlistOpen;
Purpose	Prints an HTML tag that begins a directory list. This presents information in a list of items that contain up to 20 characters. Items in this list are typically arranged in columns, typically 24 characters wide. The <LI> or htp.listItem must appear directly after you use this tag.
Parameters	none
Generates	<DIR>

htp.dirlistClose

Syntax	htp.dirlistClose;
Purpose	Prints an HTML tag that closes the directory list tag, htp.dirlistOpen.
Parameters	none
Generates	</DIR>

Character Format Tags

The character format tags are used to specify or alter the appearance of the marked text. Character format tags have opening and closing elements, and affect only the text that they surround.

Character format tags give hints to the browser as to how a character or character string should appear, but each browser determines its actual appearance. Essentially, they place text into categories such that all text in a given category is given the same special treatment, but the browser determines what that treatment is. For example, the HTML string <STRONG>Here is some text</STRONG> might appear as bold in some browsers, or might flash instead.

If a specific text attribute, such as bold is desired, a physical format tag may be necessary. See the section, "Physical Format Tags," for more information.

MAC003181

ORCL00941857

Note: All the hypertext procedures (HTP) shown in this section are also available as hypertext functions (HTF).

htp.cite

Syntax	htp.cite (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround as a citation. Usually rendered as italics.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<CITE *cattributes*>*ctext*</CITE>		

htp.code

Syntax	htp.code (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround as an example of code output. Usually rendered in monospace format, e.g. Courier.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<CODE *cattributes*>*ctext*</CODE>		

htp.emphasis

Syntax	htp.emphasis (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround as requiring typographic emphasis. This tag is equivalent to htp.em. Usually rendered as italics.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<EM *cattributes*>*ctext*</EM>		

htp.em

Alias for htp.emphasis

htp.keyboard

Syntax	htp.keyboard (ctext, cattributes);
Purpose	Prints a pair of HTML tags that specify the text they surround as text typed in by the user.,which is usually

MAC003182

ORCL00941858

rendered as monospace. This tag is equivalent to htp.kbd.

Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<KBD *cattributes*>*ctext*</KBD>		

htp.kbd Alias for htp.keyboard

htp.sample

Syntax	htp.sample (ctext, cattributes);
Purpose	Prints a pair of HTML tags that specify the text they surround as a sequence of literal characters that must be typed in the exact sequence in which they appear. Usually rendered as monospace font.

Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<SAMP *cattributes*>*ctext*</SAMP>		

htp.strong

Syntax	htp.strong (ctext, cattributes);
Purpose	Prints a pair of HTML tags that specify the text they surround as needing strong typographic emphasis. Usually rendered as bold.

Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<STRONG *cattributes*>*ctext*</STRONG>		

htp.variable

Syntax	htp.variable (ctext, cattributes);
Purpose	Prints a pair of HTML tags that specify the text they surround as a variable name, or a variable that might be entered by the user. Usually rendered as italics.

Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<VAR *cattributes*>*ctext*</VAR>		

Physical Format Tags

The physical format tags are used to specify the format of the marked text.

> Note: All the hypertext procedures (HTP) shown in this section are also available as hypertext functions (HTF).

htp.bold

Syntax	htp.bold (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround is to be rendered as boldface.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<B *cattributes*>*ctext*</B>		

htp.italic

Syntax	htp.italic (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround is to be rendered as italics.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<I *cattributes*>*ctext*</I>		

htp.teletype

Syntax	htp.teletype (ctext, cattributes);		
Purpose	Prints a pair of HTML tags that specify the text they surround is to be rendered in a fixed width typewriter font, e.g. Courier.		
Parameters	ctext	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
Generates	<TT *cattributes*>*ctext*</TT>		

MAC003184

ORCL00941860

Form Tags

The form tags are used to create and manipulate an HTML form. Forms are used to allow interactive data exchange between a Web Browser and a CGI program.

Forms can have the following types of elements:

- Input: used for a large variety of types of input fields, for example:
  - single line text
  - single line password fields
  - checkboxes
  - radio buttons
  - submit buttons
- Text area: used to create a multi–line input field.
- Select: used to allow the user to chose one or more of a set of alternatives described by textual labels. Usually rendered as a pulldown, pop up, or a fixed size list.

Note: All the hypertext procedures (HTP) shown in this section are also available as hypertext functions (HTF).

htp.formOpen

Syntax	htp.formOpen (curl, cmethod);
Purpose	Prints an HTML tag that starts the form. The curl value is required and is the URL of the CGI script, normally owa, to which the contents of the Form will be sent. The method is either "GET" or "POST."
Parameters	curl in varchar2 cmethod in varchar2 DEFAULT 'POST'
Generates	<FORM ACTION="*curl*" METHOD="*cmethod*">

htp.formClose

Syntax	htp.formClose;
Purpose	Prints an HTML tag that closes the <FORM> tag
Parameters	none
Generates	</FORM>

MAC003185

ORCL00941861

htp.formCheckbox

Syntax	htp.formCheckbox (cname, cvalue, cchecked, cattributes);
Purpose	Prints an HTML tag that inserts a checkbox which the user can toggle off or on.
Parameters	cname in varchar2 cvalue in varchar2 DEFAULT 'on' cchecked in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="checkbox" NAME="*cname*" VALUE="*cvalue*" CHECKED *cattributes*>

htp.formHidden

Syntax	htp.formHidden (cname, cvalue, cattributes);
Purpose	Prints an HTML tag that sends the content of a field along with a submitted form. The field is not visible to the end user.
Parameters	cname in varchar2 cvalue in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="hidden" NAME="*cname*" VALUE="*cvalue*" *cattributes*>

htp.formImage

Syntax	htp.formImage (cname, csrc, calign, cattributes);
Purpose	Prints an HTML tag that creates an image field that can be clicked on causing the Form to be immediately submitted. The coordinates of the selected point are measured in pixels, and returned (along with other contents of the form) in two name/ value pairs. The x–coordinate is submitted under the name of the field with ".x" appended, and the y–coordinate with the ".y" appended. Any value attribute is ignored. The image itself is specified by the CSRC attribute.
Parameters	cname in varchar2 csrc in varchar2 calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL;

Generates	<INPUT TYPE="image" NAME="*cname*" SRC="*csrc*" ALIGN="*calign*" *cattributes*>

htp.formPassword

Syntax	htp.formPassword (cname, csize, cmaxlength, cvalue, cattributes);
Purpose	Prints an HTML tag that creates a single line text entry field. Text will not be displayed as it is entered. When the user enters a password, characters are represented by asterisks on single line text entry field.
Parameters	cname in varchar2 csize in varchar2 cmaxlength in varchar2 DEFAULT NULL cvalue in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="password" NAME="*cname*" SIZE="*csize*" MAXLENGTH="*cmaxlength*" VALUE="*cvalue*" *cattributes*>

htp.formRadio

Syntax	htp.formRadio (cname, cvalue, cchecked, cattributes);
Purpose	Prints an HTML tag that inserts a radio button on the HTML Form. Used to create a set of radio buttons, each representing a different value, only one of which will be toggled on by the user. Each radio button field should have the same name. Only the selected radio button will generate a name/ value pair in submitted data area. This will require an explicit VALUE attribute.
Parameters	cname in varchar2 cvalue in varchar2 cchecked in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="radio" NAME="*cname*" VALUE="*cvalue*" CHECKED *cattributes*>

htp.formReset

Syntax	htp.formReset (cvalue, cattributes);

MAC003187

ORCL00941863

Purpose	Prints an HTML tag that creates a RESET button that, when selected, resets all the form fields to their initial values.	
Parameters	cvalue	in varchar2 DEFAULT 'Reset'
	cattributes	in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="*reset*" VALUE="*cvalue*" *cattributes*>	

htp.formSubmit

Syntax	htp.formSubmit (cname, cvalue, cattributes);	
Purpose	Prints an HTML tag that creates a button that, when selected, submits the form. If a SUBMIT button is selected to submit the Form, and that button has a name attribute specified, the submit button then contributes a name/ value pair to the submitted data.	
Parameters	cname	in varchar2
	cvalue	in varchar2 DEFAULT 'Submit'
	cattributes	in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="submit" NAME="*cname*" VALUE="*cvalue*" *cattributes*>	

htp.formText

Syntax	htp.formText (cname, csize, cmaxlength, cvalue, cattributes);	
Purpose	Prints an HTML tag that creates a field for a single line of text.	
Parameters	cname	in varchar2
	csize	in varchar2 DEFAULT NULL
	cmaxlength	in varchar2 DEFAULT NULL
	cvalue	in varchar2 DEFAULT NULL
	cattributes	in varchar2 DEFAULT NULL
Generates	<INPUT TYPE="text" NAME="*cname*" SIZE="*csize*" MAXLENGTH="*cmaxlength*" VALUE="*cvalue*" *cattributes*>	

htp.formSelectOpen

Syntax	htp.formSelectOpen (cname, cprompt, nsize, cattributes);

MAC003188

ORCL00941864

Purpose	Prints an HTML tag that begins a Select list of alternatives. Contains the attribute NAME which specifies the name that will be submitted as a name/ value pair.
Parameters	cname in varchar2 cprompt in varchar2 DEFAULT NULL nsize in integer DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	*cprompt* <SELECT NAME="cname" SIZE="nsize" *cattributes*>

Example

```
htp.formSelectOpen('greatest_player',
        'Pick the greatest player:');
htp.formSelectOption('Messier');
htp.formSelectOption('Howe');
htp.formSelectOption('Hull');.
htp.formSelectOption('Gretzky');.
htp.formSelectClose;

generates:

Pick the greatest player:
 <SELECT NAME="great player">
 <OPTION>Messier
 <OPTION>Howe
 <OPTION>Hull
 <OPTION>Gretzky
</SELECT>
```

Note: See htp.formSelectOption and htp.formSelectClose.

htp.formSelectOption

Syntax	htp.formSelectOption (cvalue, cselected, cattributes);
Purpose	Prints an HTML tag that represents one choice in the Select element.
Parameters	cvalue in varchar2 cselected in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<OPTION SELECTED *cattributes*>*cvalue* See example for htp.formSelectOpen.
Example	See htp.formSelectOpen.

MAC003189

ORCL00941865

htp.formSelectClose

Syntax	htp.formSelectClose;
Purpose	Prints an HTML tag that ends a Select list of alternatives.
Parameters	none
Generates	</SELECT>
Example	See htp.formSelectOpen.

htp.formTextarea

Syntax	htp.formTextarea (cname, nrows, ncolumns, calign, cattributes);
Purpose	Prints an HTML tag that creates a text field that has no predefined text in the text area. Used to enable the user to enter several lines of text.
Parameters	cname in varchar2 nrows in integer ncolumns in integer calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<TEXTAREA NAME="*cname*" ROWS="*nrows*" COLS="*ncolumns*" ALIGN="*calign*" *cattributes*></TEXTAREA>

htp.formTextareaOpen

Syntax	htp.formTextareaOpen (cname, nrows, ncolumns, calign, cattributes);
Purpose	Prints an HTML tag that opens a text area where you can insert predefined text that will always appear in the text field.
Parameters	cname in varchar2 nrows in integer ncolumns in integer calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<TEXTAREA NAME="*cname*" ROWS="*nrows*" COLS="*ncolumns*" ALIGN="*calign*" *cattributes*>

MAC003190

ORCL00941866

htp.formTextareaClose

Syntax	htp.formTextareaClose;
Purpose	Prints an HTML tag that ends TextArea field
Parameters	none
Generates	</TEXTAREA>

Table Tags

The Table tags allow the user to insert tables and manipulate the size and columns of the table in a document.

Note: All the hypertext procedures (HTP) shown in this section are also available as hypertext functions (HTF).

htp.tableOpen

Syntax	htp.tableOpen (cborder, calign, cnowrap, cclear, cattributes);
Purpose	Prints an HTML tag that begins an HTML table.
Parameters	cborder in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cclear in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL;
Generates	<TABLE BORDER NOWRAP ALIGN="*calign*" CLEAR="*cclear*" *cattributes*>

Note: The BORDER attribute is only determined by whether cborder is null or not null.

htp.tableClose

Syntax	htp.tableClose;
Purpose	Prints an HTML tag that ends an HTML table.
Parameters	none
Generates	</TABLE>

MAC003191

ORCL00941867

htp.tableCaption

Syntax	htp.tableCaption (ccaption, calign, cattributes);
Purpose	Prints an HTML tag that places a caption in the inserted table.
Parameters	ccaption in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<CAPTION ALIGN="*calign*" *cattributes*>*ccaption*</CAPTION>

htp.tableRowOpen

Syntax	htp.tableRowOpen (calign, cvalign,cdp, cnowrap, cattributes);
Purpose	Prints an HTML tag that inserts a row tag into a table.
Parameters	calign in varchar2 DEFAULT NULL cvalign in varchar2 DEFAULT NULL cdp in varchar2 DEFAULT NULL cnowrap in varchar2 DEFAULT NULL cattributes in varchar2 DEFAULT NULL
Generates	<TR ALIGN="*calign*" VALIGN="*cvalign*" DP="*cdp*" NOWRAP *cattributes*>

htp.tableRowClose

Syntax	htp.tableRowClose;
Purpose	Prints an HTML tag that ends a row in a table.
Parameters	none
Generates	</TR>

htp.tableHeader

Syntax	htp.tableHeader (cvalue, calign, cdp, cnowrap, crowspan, ccolspan, cattributes);
Purpose	Prints an HTML tag that inserts a table header.
Parameters	cvalue in varchar2 DEFAULT NULL calign in varchar2 DEFAULT NULL cdp in varchar2 DEFAULT NULL

MAC003192

ORCL00941868

cnowrap	in varchar2	DEFAULT NULL
crowspan	in varchar2	DEFAULT NULL
ccolspan	in varchar2	DEFAULT NULL
cattributes	in varchar2	DEFAULT NULL

Generates <TH ALIGN="*calign*" DP="*cdp*" ROWSPAN="*crowspan*" COLSPAN="*ccolspan*" NOWRAP *cattributes*>*cvalue*</TH>

htp.tableData

Syntax: htp.tableData (cvalue, calign, cdp, cnowrap, cattributes);

Purpose: Prints an HTML tag that inserts data into the rows and columns of a selected table.

Parameters:

cvalue	in varchar2	DEFAULT NULL
calign	in varchar2	DEFAULT NULL
cdp	in varchar2	DEFAULT NULL
cnowrap	in varchar2	DEFAULT NULL
cattributes	in varchar2	DEFAULT NULL

Generates <TD ALIGN="*calign*" DP="*cdp*" ROWSPAN="*crowspan*" COLSPAN="*ccolspan*" NOWRAP *cattributes*>*cvalue*</TD>

OWA_UTIL Package

The OWA_UTIL Package is a set of useful utility procedures built on top of hypertext functions and hypertext procedures.

owa_util.signature

Syntax: owa_util.signature;

Purpose: Prints an HTML line followed by a signature line on the HTML document.

Parameters: none

Generates: Prints a signature line in the HTML document that might look like the following:
"This page was produced by the Oracle Web Agent on August 9, 1995 09:30"

MAC003193

ORCL00941869

owa_util.signature (cname)

Syntax	owa_util.signature (cname);
Purpose	Allows the programmer to create a signature line on the bottom of the HTML document that has a hypertext link to view the PL/ SQL source for that procedure.
Parameters	cname in varchar2
Generates	Prints a signature line on the bottom of the HTML document that has a link to the actual PL/ SQL source for that procedure. The link calls the procedure showsource. The line would look like the following: "This page was produced by the Oracle Web Agent on 6/ 14/ 95 09:30" View PL/SQL Source

owa_util.showsource (cname)

Syntax	owa_util.showsource (cname);
Purpose	Prints the source of the specified PL/ SQL procedure, function, or package. If a procedure or function which belongs to a package is specified, then the entire package is displayed.
Parameters	cname in varchar2
Generates	Generates the source code of the specified PL/ SQL procedure.

owa_util.showpage

Syntax	owa_util.showpage;
Purpose	This procedure allows a user to view the HTML output of a PL/ SQL procedure call from SQL*Plus, SQL*DBA, or Oracle Server Manager. The PL/ SQL procedure must use HTP and/ or HTF to generate the HTML page, and owa_util.showpage must be issued after the PL/ SQL procedure has been called and before any other HTP or HTF subprograms are directly or indirectly called. This method is useful for generating pages filled with static data.

MAC003194

ORCL00941870

Note that this procedure uses dbms_output and thus is limited to 255 characters per line and an overall buffer size of 1,000,000 bytes.

Parameters: none

Generates: One can use this procedure to generate static pages in SQL*Plus that can then be accessed as a standard HTML page. For example:

```
SQL>set serveroutput on
SQL>spool gretzky.html
SQL>execute hockey.pass('Gretzky")
SQL>execute owa_util.showpage
SQL>exit
```

This would generate an HTML page which could be accessed from Web clients.

owa_util.get_cgi_env(function)

Syntax: owa_util.get_cgi_env(param_name in varchar2);

Purpose: Allows programmer to retrieve the value of the specified CGI environment variable in the PL/ SQL procedure.

Note that param_name is case–insensitive.

Parameters: param_name in varchar2

Returns: Returns value of specified CGI environment variable for PL/ SQL procedure. If the value is not set, returns null

owa_util.print_cgi_env

Syntax: owa_util.print_cgi_env;

Purpose: enables programmer to print all of the CGI environment variables made available by the Web Agent to the PL/ SQL procedures. This utility is good for testing purposes.

Parameters: none

Generates: Prints CGI environment variables made available by the Web Agent to the PL/ SQL procedures.

MAC003195

ORCL00941871

owa_util.mime_header

Syntax	owa_util.mime_header;
Purpose	Enables programmer to change the default MIME header that the Web Agent returns. This must come before any htp.print or htp.prn calls in order to signal the Web Agent not to use the default.
Parameters	ccontent_type in varchar2
Generates	Content–type: <ccontent_type>\ n\ n

owa_util.get_owa_service_path (Function)

Syntax	owa_util.get_owa_service_path;
Purpose	Returns the name of the currently active path with its full virtual path, plus the currently active service. For example, a call to get_owa_service_path could return / ows–bi n/ myser vi ce/ owa/ .
Parameters	none
Returns	The service path. The datatype is varchar2.

owa_util.tableprint

Syntax	owa_util.tablePrint;		
Purpose	Enables programmers to print Oracle tables as either preformatted or HTML tables, depending upon Web browser capabilities. Note that RAW COLUMNS are supported, however LONG RAW are not. References to LONG RAW columns will print the result 'Not Printable'. In this case, cat t r i but es is the second, rather than the last, parameter.		
Parameters	ctable	in varchar2	
	cattributes	in varchar2	DEFAULT NULL
	ntable_type	in integer	DEFAULT HTML_TABLE
	ccolumns	in varchar2	DEFAULT '*'
	cclauses	in varchar2	DEFAULT NULL
	ccol_aliases	in varchar2	DEFAULT NULL
	nrow_min	in number	DEFAULT 0
	nrow_max	in number	DEFAULT NULL

MAC003196

ORCL00941872

Note that ntable_type can be either owa_util.html_table or owa_util.pre_table.

Generates	Prints out either a preformatted or HTML table.
Returns	True or False as to whether there are more rows available beyond the nrow_max requested.

Example

For browsers that don't support HTML tables, create the following procedure:

```
create or replace procedure showemps is
  ignore_more boolean;
begin
  ignore_more := owa_util.tablePrint('emp', 'BORDER',
OWA_UTIL.PRE_TABLE);
end;
 and requesting a URL like this example:
http://myhost:8080/ows-bin/hr/owa/showemps

returns to the client:
<PRE>
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

```
</PRE>
```

To view just the employees in department 10, and only their employee ids, names, and salaries, create the following procedure:

```
create or replace procedure showemps_10 is
   ignore_more boolean;
begin
 ignore_more := owa_util.tablePrint
```


MAC003197

ORCL00941873

```
    ('EMP', 'BORDER', OWA_UTIL.PRE_TABLE,
        'empno, ename, sal',
        'where deptno=10 order by empno',
        'Employee Number, Name, Salary');
end;
```

A request for a URL like
`http://myhost:8080/ows-bin/hr/owa/showemps_10` would return the following to the client:

```
<PRE>
-------------------------------------
| Employee Number |   Name  | Salary |
-------------------------------------
| 7782            | CLARK   | 2450   |
| 7839            | KING    | 5000   |
| 7934            | MILLER  | 1300   |
-------------------------------------
</PRE>
```

For browsers that do support HTML tables, to view the department table in an HTML table, create the following procedure:

```
create or replace procedure showdept is
 ignore_more boolean;
begin
   ignore_more := owa_util.tablePrint('dept', 'BORDER');
end;
```

A request for a URL like `http://myhost:8080/ows-bin/hr/owa/showdept` would return the following to the client:

```
<TABLE BORDER>
<TR>
<TH>DEPTNO</TH>
<TH>DNAME</TH>
<TH>LOC</TH>
</TR>
<TR>
<TD ALIGN="LEFT">10</TD>
<TD ALIGN="LEFT">ACCOUNTING</TD>
<TD ALIGN="LEFT">NEW YORK</TD>
</TR>
<TR>
<TD ALIGN="LEFT">20</TD>
<TD ALIGN="LEFT">RESEARCH</TD>
<TD ALIGN="LEFT">DALLAS</TD>
</TR>
<TR>
<TD ALIGN="LEFT">30</TD>
```


MAC003198

ORCL00941874

```
<TD ALIGN="LEFT">SALES</TD>
<TD ALIGN="LEFT">CHICAGO</TD>
</TR>
<TR>
<TD ALIGN="LEFT">40</TD>
<TD ALIGN="LEFT">OPERATIONS</TD>
<TD ALIGN="LEFT">BOSTON</TD>
</TR>
</TABLE>
```

which a Web browser can format to look like this:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO

Customized Extensions to HTP Packages

The design of the hypertext procedure and function packages allows you to use customized extensions. Therefore, as the HTML standard changes, you can add new functionality similar to the hypertext procedure and function packages to reflect those changes.

Here is an example of customized packages using non–standard <BLINK> and imaginary <SHOUT>tags:

```
create package nsf as
    function blink(cbuf in varchar2) return varchar2;
    function shout(cbuf in varchar2) return varchar2;
end;
create package body nsf as
    function blink(cbuf in varchar2) return varchar2 is
    begin return ('<BLINK>' || cbuf || '</BLINK>');
    end;
    function shout(cbuf in varchar2) return varchar2 is
    begin return ('<SHOUT>' || cbuf || '</SHOUT>');
    end;
end;

create package nsp as
    procedure blink(cbuf  in varchar2);
    procedure shout(cbuf  in varchar2);
end;
create package body nsp as
    procedure blink(cbuf  in varchar2) is
```


MAC003199

ORCL00941875

```
    begin htp.print(nsf.blink(cbuf)); end;
    procedure shout(cbuf  in varchar2) is
    begin htp.print(nsf.shout(cbuf)); end;
end;
```

Now you can begin to use these procedures and functions in your own procedure.

```
create procedure nonstandard as
begin
    nsp.blink('Gee this hurts my eyes!');
    htp.print('And I might ' || nsf.shout('get mad!'));
end;
```

For more examples of using the Developer's Toolkit, see the section, "Passing Parameters to PL/SQL" in Chapter 5, "Oracle Web Agent".

MAC003200

ORCL00941876

CHAPTER

7

The Oracle WebServer Administration Utility

This chapter shows you how to use the Administration Utility of the Oracle WebServer. The Administration Utility enables you to configure the components of the Oracle WebServer from any web browser, so you do not have to manually edit files to perform common operations.

To navigate to the Administration Utility, click Administration from the Oracle WebServer home page. The utility is divided into the following three areas:

- The Oracle7 Database Server. This enables you to start up or shut down an Oracle7 database.
- The Oracle Web Listener. This enables you to create an instance of the Oracle Web Listener or to modify the parameters of an existing instance.
- The Oracle Web Agent. This enables you to create or modify web agent services, including the specification of password–protected security.

The Administration page is shown in Figure 7 – 1. Note: If you are viewing this document online, there are no screen shots provided.

MAC003201

ORCL00941877

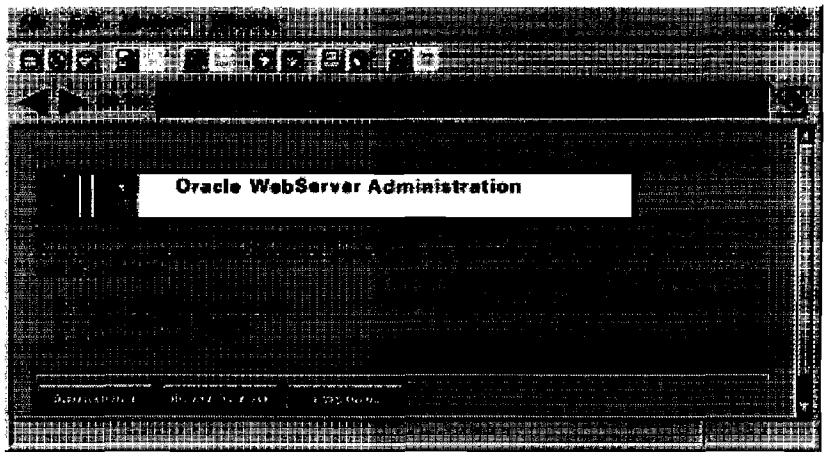


Figure 7 – 1 Administration Utility Form

Choose the area you want to use by clicking the appropriate button. This chapter contains a section for each administration area.

Database Administration

Use this page, shown in Figure 7 – 2, to start up or shut down any Oracle7 database running on the same machine as the Oracle Web Listener.

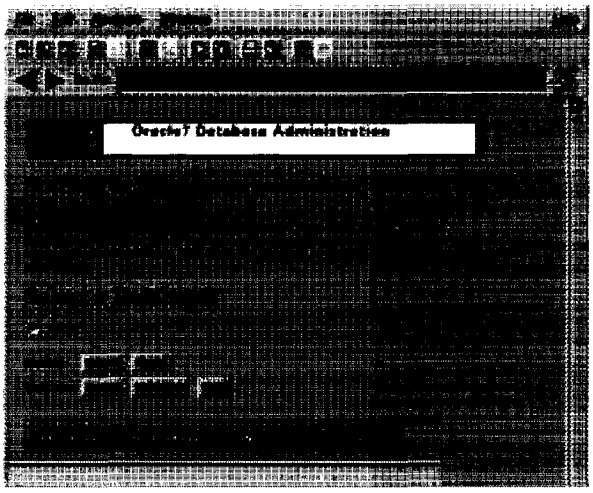


Figure 7 – 2 Database Administration Page

MAC003202

ORCL00941878

The page lists the available databases. This list comes from a file on the machine on which the Web Listener resides. On UNIX machines, this is known as the ORATAB file. See your platform–specific documentation for the location and format of this file. If no databases are listed, then the file contains no database descriptions.

The procedure is as follows:

1. Choose the database to which you want to apply the operation by clicking the radio button beside its name in the list.
2. Choose the operation you wish to perform and the mode by clicking the appropriate button. The modes are explained in the following sections.

For more information on database startup and shutdown, see the Oracle7 Server Administrator's Guide.

Startup Modes

You can perform database startup in either of the following two modes:

Normal

Starts up the database normally. This assumes that the database currently is shut down. If there are any errors during startup, they are reported, and the database is left shut down.

Force

Shuts down the database in the Abort mode (as explained under Shutdown Modes below), and then starts it up in the Normal mode.

Shutdown Modes

You can perform database shutdown in any of the following three modes:

Normal

Shuts down the database cleanly, waiting for current session to disconnect normally. New connections are not allowed while shutdown is in progress.

MAC003203

ORCL00941879

Immediate

Attempts to shut the database down immediately and cleanly. Disconnect any existing user sessions, but wait for active calls to complete before disconnecting. Roll back all uncommitted transactions .

Abort

Shuts the database down as quickly as possible, killing all active Oracle7 processes and freeing the memory used. This leaves current transactions unresolved. When you start up the database again, instance recovery is performed automatically. This option is recommended only for emergency use.

For more information on Oracle7 databases including startup and shutdown, see the Oracle7 Server Concepts Manual and the Oracle7 Server Administrator's Guide.

Web Agent Administration

The Web Agent Administration page enables you to create, delete, or modify Web Agent Services, which Web Agents use to connect to the database. An Oracle7 Web Agent Service specifies values needed by the Web Agent every time it makes a database connection. For example, an Oracle7 username and password are needed, as well as an Oracle7 SID or SQL*Net V2 service name. For more information on Web Agent Services, see "Web Agent Service" in Chapter 5.

Creating or Modifying Web Agent Services

Use the Web Agent section of the Administration Utility to create a new Web Agent Service or to copy, delete, or modify an existing one. The initial Web Agent Administratin page is shown in Figure 7 – 3.

MAC003204

ORCL00941880



Figure 7 – 3 Initial Web Agent Administration Page

To create a new Oracle Web Agent Service, click Create New Service. The Oracle Web Agent Administration Creation form appears.

To modify, copy, or delete an existing Oracle Web Agent Service, click the name of that service in the list provided. The Oracle Web Agent Administration Modification form appears.

The Oracle Web Agent Administration Service Creation Form

Use this fill–out form, shown in Figure 7 – 4 to enter the parameters for the service you want to create. The parameters are explained below.

MAC003205

ORCL00941881

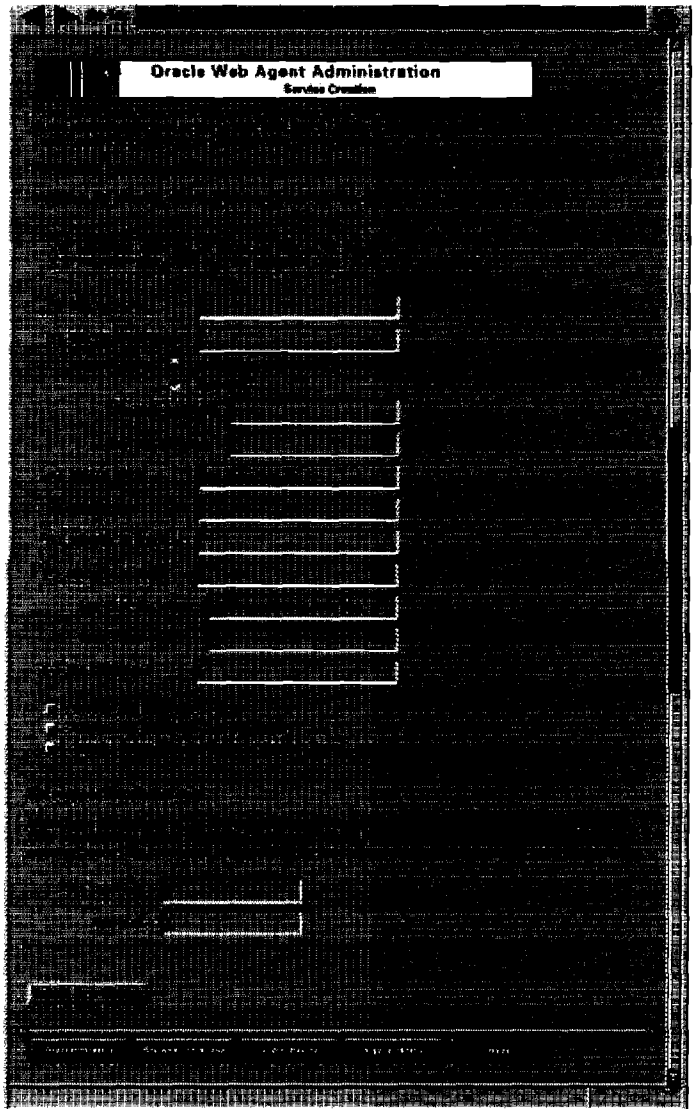


Figure 7 – 4 Create Web Agent Service Form

OWA Service:

The name of the service you want to create. The name is not case sensitive and must be unique.

MAC003206

ORCL00941882

OWA Database User:

The name of the Oracle7 user that will be used to access the database for this service. If operating system (OS) authentication is used, then the Web Listener must be run with the same effective user as the Oracle7 user. For more information on Oracle7 users, see "CREATE USER" in the Oracle7 Server SQL Reference.

Identified By

A radio button that determines whether Oracle7 or the Operating System performs authentication.

OWA User Password:

The Oracle7 password for the associated OWA Database User. If OS authentication is be used for the OWA Database User, as determined by the Identified By radio button, you can leave this field blank. Otherwise, the username and password for the OWA Database User are verified by Oracle7 upon submission of a Web Agent service.

If an error occurs when you attempt to submit this form, the password fields are blanked out and must be filled in again. This is to provide the tightest security currently available for your passwords. As a short–cut, you can select the Back button on your Web Browser to return to, and correct, the original submission form.

Confirm Password:

This field is used to verify the OWA user password you entered above before recording it.

ORACLE_HOME:

The location of the Oracle7 code tree in the file system. This should be the ORACLE_HOME for the database to which this Web Agent service connects, unless the Web Agent service is set up to connect to a remote database (over SQL*Net). In that case, specify the ORACLE_HOME where the Web Agent is installed.

ORACLE_SID:

The system ID of the database. You only need this if the Web Agent is to connect to a local database. For more on Oracle7 SIDs, see the Oracle7 Server Administrator's Guide.

MAC003207

ORCL00941883

SQL*Net V2 Service:

The SQL*Net service name that indicates which database to connect to. If the database is on the same machine as the Web Listener, then the SQL*Net V2 Service field can be left empty, and an Oracle7 SID specified instead. If both parameters are specified, the SQL*Net V2 Service takes precedence. If SQL*Net V1 is installed on this machine, then a SQL*Net V1 Connect String may also be specified here. For more on the SQL*Net V2 service name, see Understanding SQL*Net.

Authorized Ports:

To provide security for your Web Agent service, use the Authorized Ports field to list (space–delimited) the valid Web Listener network ports for the Web Agent to service. This field is required, and the network ports must be listed explicitly.

Log File Directory:

The directory where the Oracle Web Agent writes its error file. The error file will have the name service_name.err.

If the Web Agent is unable to resolve a service name, and no service called OWA_DEFAULT_SERVICE exists, then it will write error and logging information to the file owa.err.

The default value for the Log File Directory is the log directory under the OWS directory within ORACLE_HOME.

Note that these log files are different from those used by the Oracle7 Server for database recovery. The latter are explained in the Oracle7 Server Concepts Manual.

HTML Error Page :

If the Oracle Web Agent is unable to process a request, it returns this HTML page to the user making the request, and at the same time writes detailed errors and debugging information to the error message file in the Log File Directory. If this field is left blank, a default error message is returned to the client.

NLS Language :

This field specifies the NLS_LANG parameter, which specifies the character set to be used. If this field is left blank, then the Web Agent administration program looks it up in the configuration file for the

MAC003208

ORCL00941884

database upon service submission.

Create OWA Database User :

Create an Oracle7 user using the OWA user name and OWA user password provided. If the Identified by ... Operating System radio button is selected, then the user is created as identified externally, which means that the operating system, rather than Oracle, performs user authentication. In that case, this value is ignored.

Change OWA Database User Password:

Change the Oracle7 Database user's existing password to that specified by the OWA User Password. If the Identified by ... Operating System radio button is selected, then the user will be altered to be identified externally, which means that the operating system, rather than Oracle, performs user authentication.

Install Oracle WebServer Developer's Toolkit PL/SQL Packages

Create the Oracle WebServer Developer's Toolkit PL/ SQL packages—OWA, HTP, HTF, and OWA_UTIL—under the schema of the OWA Database User.

After filling in the above fields, click the Submit New Service button to create the service. To activate your new service, go to the Oracle WebServer Administration form and click Configure for the Web Listener you want to associate with this agent. Then go to "Directory Mappings" to create a virtual path, with the service name as the last part of the path, to the OWS/ bin directory. Virtual paths are discussed under "Directory Mappings", later in this chapter.

The Oracle Web Agent Administration Service Modification Form

This form contains the same fields as the Oracle Web Agent Administration Creation Form. The fields show their current values except for the password fields, which are left blank for security reasons. You can leave these fields blank unless you want to change the password.

MAC003209

ORCL00941885

At the bottom of the form are the following three buttons:

Copy Service

This creates a copy of the service, leaving the original intact. To do this, simply change the service name (the OWA service field) and any other parameters you wish to alter for the new copy and click this button. If you specify no new password, the original password is copied. After clicking this button, you have two services.

Modify Service

This changes the existing service. You can change the name or leave it intact as desired. Make the desired changes and click the button. After clicking this button, you have one service.

Delete Service

This destroys the service. For this purpose, it does not matter whether you change any parameters. After clicking this button, you have no service. Calls to the deleted service in existing Web pages or PL/ SQL blocks do not produce errors, however. Instead, they are reinterpreted as calls to OWA_DEFAULT_SERVICE.

Securing a Web Agent Service

The mechanism of "services" provides a security framework such that individual Web Agent services can be password protected by the Web Listener. For example, to password protect the hr service, you would do the following:

1. Create a virtual path for the hr service by specifying a Web Listener directory mapping. This mapping must use a value of 'C' to indicate an executable. For more information on setting the directory mappings, see Directory Mappings later in this chapter.
2. Create a realm consisting of those users and groups (UNIX only) who are to have access to the service. For instruction on how to do this, see Specifying the Privileged Users, later in this chapter.
3. Assign the realm to the service and specify either Basic or Digest Authentication. To learn how to perform the assignment, see Performing the Security Assignments, later in this chapter. To find out about Basic and Digest Authentication, see Security Schemes, later in this chapter.

MAC003210

ORCL00941886

For more information of Web Listener security in general, see "Security Definitions", later in this chapter, and "Client to Listener Security" in Chapter 3.

Web Listener Administration

The Oracle Web Listener is a reliable, high–performance HTTP server suitable for high–traffic web applications where quick response is crucial.

From the Web Listener Administration page, shown in Figure 7 – 5, you can move to other pages that provide a convenient way for you to do the following:

- Create a new Web Listener
- Change or copy an existing Web Listener
- Start a Web Listener
- Stop a Web Listener
- See whether an Oracle Web Listener is currently running.

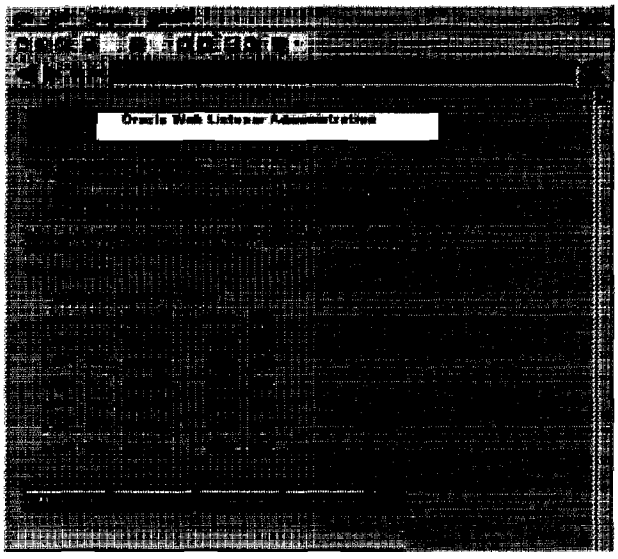


Figure 7 – 5 Web Listener Administration Page

To create a new Oracle Web Listener, click Create New Oracle Web Listener. This takes you to the Listener Creation form.

MAC003211

ORCL00941887

To perform any of the other operations, choose the appropriate existing Web Listener from the list provided. This list uniquely identifies each Web Listener by the network port on which it is configured to listen. For your convenience, the directory in which the configuration file resides is specified beside the port number. Unlike the port number, this directory need not be unique.

If a Web Listener is running, the word "running" appears beside it in the list.

To modify an existing Oracle Web Listener, select it in the list and click Configure. This takes you to the Configure Extended Parameters form, with the current parameter values filled in. Effect the changes by writing over the current values. For information on the parameters themselves, see "Configure Extended Parameters," later in this chapter.

To start a currently–stopped Oracle Web Listener, select it in the list and click Start.

To stop a currently–running Oracle Web Listener, select it in the list and click Stop.

> Note: You might need to perform a manual startup of an existing Web Listener if you inadvertently shut down the Web Listener that the Administration Utility itself is using. To find out how to do this, see "The WLCTL" in Chapter 3.

The Oracle Web Listener Creation Form

Use this form, shown in the next two figures, to create a new Oracle Web Listener by filling in the listed parameters. These are the parameters the Web Listener requires in order to run. After filling in all required parameters, press the appropriate button to choose a Basic Configuration or an Advanced Configuration. In either case, the Web Listener is created. If you choose Basic Configuration, the parameters you fill in here are all that are needed, and you can start the Web Listener you have created. If you choose Advanced Configuration, you go to the Configure Extended Parameters form to provide optional information. This is equivalent to selecting an existing Web Listener from the list in the initial Web Listener Administration page and clicking Configure. For more information on the Configure Extended Parameters form, see "The Configure Extended Parameters Form" later in this chapter.

MAC003212

ORCL00941888

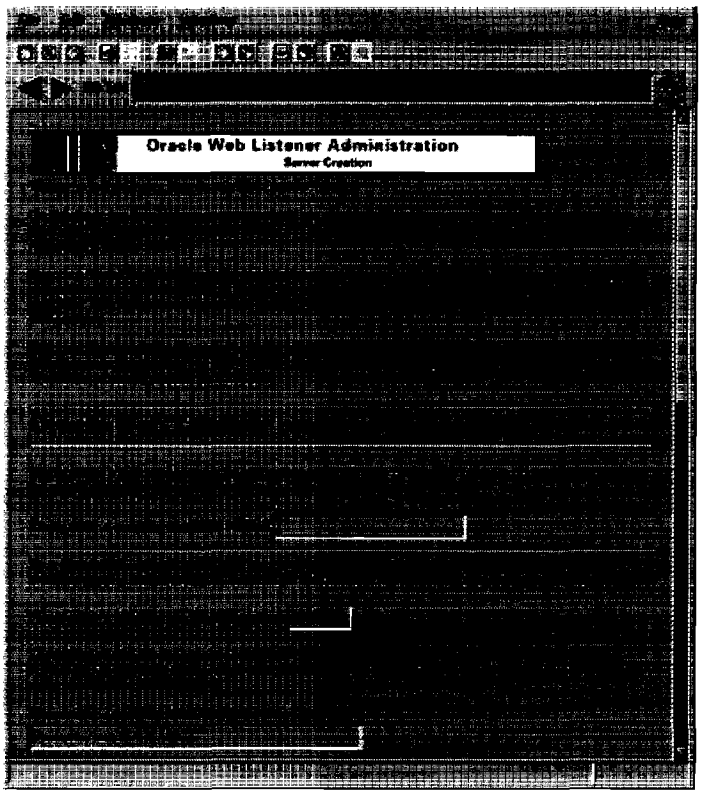


Figure 7 – 6 Listener Creation Form, Part 1

The Basic Configuration Parameters

The parameters required for a basic configuration are explained below:

Host Name

This is the name of the machine on which the Oracle Web Listener is to reside. It can either be an actual machine name or an alias in the form of a domain name, such as:

www. oracl e. com

Port Number

The Oracle Web Listener listens for network messages on a particular port. Use this parameter to specify that port. Numbers can range from 0 to 65535 with numbers below 1024 requiring superuser privileges on UNIX systems.

MAC003213

ORCL00941889

Configuration Directory

This is the directory where the configuration file for the Web Listener is to be located. A default is provided.

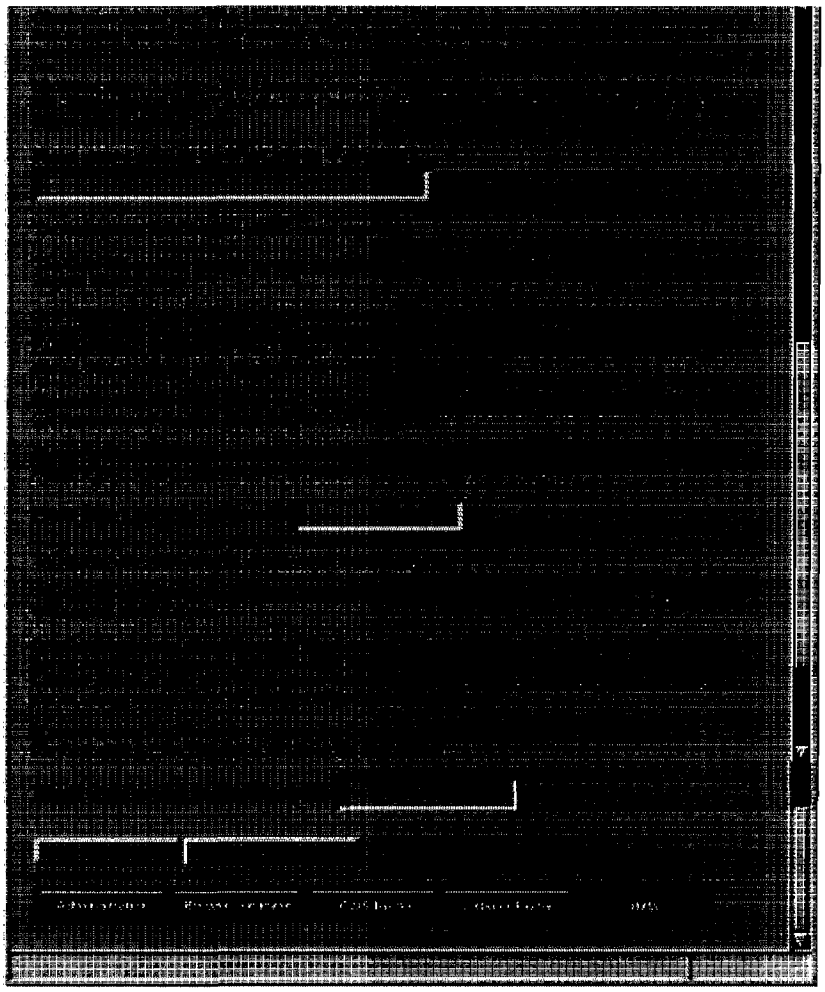


Figure 7 – 7 Listener Creation Form, Part 2

Web Listener Root (/) Directory

This is the directory that serves as the root (/) directory for the Web Listener. This is often called the "Document Root". It is used to resolve client calls that do not specify a directory. Thus a URL that does not contain a directory, such as http:/ / www.oracle.com/ index.html,

MAC003214

ORCL00941890

causes the Web Listener to retrieve the document "index.html" from the directory specified here.

User ID (UNIX only)

The User ID specifies the user (or user identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this user or by root with "Set user ID" set. The User ID can be either a username or a user ID number. The default is the effective ID. Specifying the Oracle owner or an Oracle user with the DBA role is not advised, as this compromises database security. For more information, see "roles" and "Database Administrators" in the Oracle7 Server Concepts Manual.

Group ID (UNIX only)

The Group ID specifies the group (or group identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this group or by root with "Set group ID" set. The Group ID can be either a group name or a GID number. The default is the effective GID.

Note that any Oracle Web Listener that invokes the Oracle Web Agent must have read access to the OWA configuration file (owa.cfg). However, since this file contains Oracle usernames and their corresponding passwords, it is recommended that read permissions only be given to the oracle owner and to one privileged group. This is the group that a Web Listener should run under if it is going to run the Oracle Web Agent.

A convenient group to use for this is the OracleDBA group. However, if you run an Oracle Web Listener under the DBA group, you will want to place strict controls on who can create CGI programs that will be invoked by that Web Listener. This is because the CGI program will run with group DBA privileges, and someone with malicious intent could potentially startup and shutdown your Oracle database, or even remove Oracle files. If you wish to avoid this potential security risk, then you need to do the following:

1. Create a new operating system group.
2. Make the Oracle owner a member of the group created in step 1.
3. Log in as the Oracle owner and change the group ownership of owa.cfg to the group created in step 1.

   Note: If at any future point in time you remove the owa.cfg file, you must reset its group ownership when it is recreated.

MAC003215

ORCL00941891

4. Configure the Group ID parameter of your Oracle Web Listener to be the name of the group created in step 1.
5. Restart your Administration Server (the Web Listener that the Administration Utility is using). You must do this to ensure that the Web Agent Administration program inherits the group created in step 1. Do not change the Group ID for the administration server. It must continue to run as the OracleDBA group.

Starting a New Oracle Web Listener

Once you have create a Web Listener, you are shown a Success! message. At this point, you are ready to start your new Oracle Web Listener. Ordinarily, you do this by navigating to the Oracle Web Listener Administration page and clicking start. Note that most web browsers buffer visited pages, and therefore do not reload them when you return to them using something like a Back button. For the new Web Listener to be listed as available for startup, you must navigate to it in such a way that the page is reloaded, for example by reentering the URL.

Reinitializing the Oracle Web Listener

If you selected Advanced Configuration and all the parameters you entered were valid, you will be shown a Success! message. You will then be presented with the Web Listener Configure Extended Parameters form which allows you to modify any of the Oracle Web Listener configuration parameters. It is important to remember that once the Web Listener is started, however, it does not automatically recognize new files added to its virtual file system or parameters changed in its configuration file. If you would like to add files to the Web Listener, you should navigate to the Web Listener Administration page and click Configure next to the Web Listener you want to modify.

The Configure Extended Parameters Form

This form provides a convenient way to customize an existing Oracle Web Listener for your particular environment. By using this form, you can avoid having to manually edit the Web Listener configuration file, thus eliminating the need to be familiar with the exact syntax for each parameter.

From this form, you will be able to customize the following:

- Network Parameters
- Log File Parameters

MAC003216

ORCL00941892

- Miscellaneous Web Listener Parameters
- User and Group Parameters
- Directory Mappings
- File Cache Definitions
- Language Extensions
- MIME Types
- Encoding Extensions
- Security Definitions for files and directories.

To go to the form for any of these sets of parameters, click it in the list. Alternatively, you can scroll through the Configure Extended Parameters Form to visit each parameter set in sequence except Security Definitions, which you can reach only by clicking.

These parameter sets are explained later in this chapter. For more information, see "Configuration Parameters" in Chapter 3.

Below each parameter set are three buttons. Click one to apply the changes you have made. The buttons are as follows:

Copy Web Listener

If you select this button, you must first change the Port Number listed in the Network Parameters section on this form. Then, the entire contents of this form will be used to create a new configuration file and thus a new Oracle Web Listener, leaving the existing Web Listener's configuration file intact. If you reached this form as part of a "create" operation, two Web Listeners are created.

Modify Web Listener

Pushing this button will overwrite the existing Oracle Web Listener's configuration file. This operation creates a new Web Listener and deletes the existing one. If you reached this form as part of a "create" operation, one Web Listener is created.

Delete Web Listener

Pushing this button will bring up a dialog for you to confirm the deletion. Confirming will do the following:

- Shut down the specified Oracle Web Listener if is currently running.

MAC003217

ORCL00941893

- Delete the specified Oracle Web Listener's configuration entry.
- Delete the specified Oracle Web Listener's configuration file.
- If you reached this form as part of a "create" operation, abort the create.

Network Parameters

This section defines the following parameters:

Host Name

The Host Name is the name of the machine on which the Web Listener executes. You must provide this parameter if the Web Listener cannot determine the host name or if the Web Listener executes under an alias. The Web Listener needs this information for logging purposes and to answer SNMP (Simple Network Management Protocol, an Internet standard protocol for checking the status of network components) requests for status. If no host name is specified here, the Web Listener uses the primary Internet address as the default.

Host Address

The Host Address is the IP address on which the Web Listener accepts connections. Specifying this address enables a Web Listener residing on a multi–homed machine to designate explicitly the direction from which it receives connections. Multi–homed machines are machines that are configured to have more than one IP address. Specifying ANY, which is the default, indicates that the Web Listener is willing to accept connections arriving on any of its available IP addresses. Leave the parameter set to ANY unless you have a specific reason for changing it.

Port Number

The Port Number is the advertised port on which the Web Listener listens for connections. Except for the value Port Number = 0, any port number is valid provided the Web Listener has the authority to use the requested port. Port numbers between 0 and 1023 are reserved ports and require root (or special) privileges in order to be used. Non–privileged processes can use any port greater than or equal to 1024 unless those ports are reserved for some other use. Ordinarily, the Web Listener uses 80 as the default port if no port number is specified

MAC003218

ORCL00941894

Max Connect Count

The Maximum Connect Count is the maximum number of simultaneous connections to the Web Listener that the Web Listener will accept. After the maximum count is reached, the Web Listener ignores requests for new connections. The default value is 50. The maximum and optimal value is 338.

DNS Resolution

Domain Name Service Resolution indicates whether and when the Web Listener will resolve hostnames from IP addresses. If the value of this parameter is set to ALWAYS, then the Web Listener always resolves the name. If it is set to LAZY, then the Web Listener resolves the name only on demand. If it is set to LAZY_WITH_CGI, then the Web Listener also resolves the name on demand for CGI applications. If it is set to NEVER, then the Web Listener never resolves the name, even on demand. The default is NEVER.

Web Listener PID

The Web Listener PID file is a one–line file that contains the Web Listener's process identifier. There is no default. This information can be used from a UNIX shell command line to stop the Web Listener, determine its status, or reinitialize a Web Listener that is already running. Normally, however, you use the Administration Utility itself to do these things.

Log File Parameters

The following parameters are defined in this section:

Log File Directory

The Log File Directory determines the directory where error and information logs are kept. You need to make sure that this directory actually exists, because the Web Listener will not create one if it does not. The Web Listener needs write permission on this directory and on the files it contains. There is no default.

Info File Name

The Log Information File contains periodic and event–driven messages that the Web Listener generates to help WebServer Administrators monitor their web site. The Oracle Web Listener records connections in the industry–standard Common Log Format. This format includes the

MAC003219

ORCL00941895

first line of the client's request, which specifies the action, URL, and protocol version. It also includes such information as the client's host address, whether the request was successful, and how many bytes were transmitted. This information may be used later to analyze what types of accesses are performed by which users, and to check for errors.

Error File Name

The Log Error File contains event–driven error messages that the Web Listener generates as it encounters problems with clients, resource access, and various other internal problems. There is no default.

Miscellaneous Web Listener Parameters

The following parameters are defined in this section:

Configuration Directory

This is the directory where the configuration file for the Web Listener is to be located. The current value is shown. If you change this value to a new directory, the file containing the changes you currently are making to all Web Listener parameters will be written to this new directory.

Initial File

The Initial File specifies the file that the Web Listener looks for when it is sent a directory URL. For example, if the Initial file is welcome, a request for http:/ / www/ stuff causes the Web Listener to return the file / stuff/ welcome. The default is initial.

User Directory

The User Directory is applicable only to Web Listeners running under the UNIX Operating System. It specifies the subdirectory in a user's directory the Web Listener searches when it receives a URL that starts with / ~username/ . For example, if at www, this value is set to public_html, then a request for http:/ / www/ ~charles/ initial.htm causes the Web Listener to return the file ~charles/ public_html/ initial.htm. There is no default.

User Directory Initial File

The User Directory Initial File is a special case. For directories accessed with a URL with / ~username/ , the Oracle Web Listener will not use HTTP incomplete file negotiation and so the Web Listener needs a

MAC003220

ORCL00941896

complete filename in the URL. The default is initial.htm.

Default MIME Type

The default MIME (Multimedia Internet Mail Extension) Type is the MIME type for the Web Listener to use if a URL uses a file name extension to request a MIME type the Web Listener does not support. MIME types specify the kind of file, such as image, audio, or video, and the subtype gives the precise file format. The default for this parameter is application/ octet–stream, which means the Web Listener will regard any unsupported MIME type as a binary file. If you would like the Web Listener to regard unsupported MIME types as anything other than application/ octet–stream, you need to set this parameter.

Default Character Set

This is the character set that the Web Listener uses if it does not recognize the character set specified in the URL's file extension. For more information, see "Language Extensions", later in this chapter.

Preferred Language

This is the language that the Web Listener uses if the specified or defaulted character set can apply to multiple languages, and the exact language is not specified. For more information, see "Language Extensions," later in this chapter.

Image Map

The contents of this field denote the extension that the Web Listener expects image maps to have. Thus, if your image maps have .map as their extension, set this parameter to map.

An image map is a graphic that is specially configured to act as a selector for one or more hyperlinks, based on where the user clicks on the graphic as displayed on the browser. Oracle WebServer processes image maps internally, without creating a separate process to handle them. An example of an image map would be a map of the world that allows the user to bring up an HTML document on a specific country by clicking on the country on the map. The Oracle Web Listener supports the same image map file formats as are read by the NCSA server.

MAC003221

ORCL00941897

Secure Information Parameters

The following parameters are applicable only to Web Listeners running on the UNIX Operating System:

User ID

The User ID specifies the user (or user identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this user or by root with "Set user ID" set. The User ID can be either a user name or a userid number. The default is the effective ID.

Group ID

The Group ID specifies the group (or group identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this group or by root with "Set group ID" set. The Group ID can be either a group name or a GID number. The default is the effective GID. For further information on the use of Group ID for security, see "Basic Configuration Parameters," earlier in this chapter.

Directory Mappings

The Directory Mappings control the mapping of directories from the file system into the Web Listener's virtual file system. The Oracle Web Listener supports the concept of a "virtual" or mapped filesystem. This feature enables the URL used to reference a set of files (directory) to be kept constant even if the files are moved. The translation of the network visible "virtual" path to an actual path is performed at runtime based on the mappings set in this section.

You specify one mapping per line, putting the actual path on the left, the virtual path on the right, and selecting from the pop–up menu one of the following:

- 'C', which indicates that the directory contains CGI or other scripts. Although the directory can contain multiple CGI scripts, it cannot be recursive. That is to say, none of the CGI scripts can be contained in subdirectories.
- 'R', which indicates that the subdirectories should be mapped recursively. That is to say, subdirectories of the current directory are included.

MAC003222

ORCL00941898

- 'N', which indicates that the subdirectories should not be mapped recursively. That is to say, subdirectories of the current directory are not included.

Although you can list parameters in most sections of the configuration file in any order, you must order the parameters in this section so that parent virtual directories are listed before the virtual subdirectories they contain. Hence, the first directory mapping must reference the virtual root directory. The following is an example for UNIX systems:

File-System Directory	R/N/C	Virtual Directory
/html	R	/
/prodinfo	N	/products
/final/xr25	R	/products/xr25
/misc/people	R	/employees
/misc/routines	C	/scripts

In the example above, /html is the virtual root directory. This means that any URL that does not contain a directory gets mapped to /html. Thus, the URL http://www/foo.html will cause the Web Listener to look for foo.html in the /html directory. Since this particular mapping is set to be recursive, any URL which contains a directory other than /products, /employees, and /scripts will get mapped to /html/ directory. Thus, the URL http://www/foo/bar.html will cause the Web Listener to look for bar.html in the /html/foo directory. Since /prodinfo is set to non-recursive, any URL that contains a subdirectory of /products causes an error. The exception would be a URL that contains /products/xr25, because the third line maps this virtual directory to /final/xr25. The last line is an example of a mapping to a directory that contains CGI and other scripts. Thus, the URL http://www/scripts/baz will cause the Web Listener to execute /misc/routines/baz.

> Note: If you change the directory mappings, you must reload the Web Listener to put the new mappings into effect. To do this, use the WLCTL utility, as described under "WLCTL" in Chapter 3 of this document.

File Cache Definitions

This section designates cached files. Files listed here are kept open for the life of the Web Listener. Files not listed here are not cached, and the Web Listener releases the resources required to access them once there are no outstanding client references to them—that is, once their reference counts drop to zero.

MAC003223

ORCL00941899

File caching decreases the access time required for commonly–used files, but at the cost of increasing the memory that the WebServer process uses. For this reason, you should not cache very large files unless absolutely necessary.

You can specify files for caching in any of the following three ways:

- by giving an explicit filename
- by giving an incomplete filename that ends in an asterisk to *indicate that all files matching the portion before the asterisk are* affected
- by giving a directory, in which case all files in that directory and all of its subdirectories are affected

File references must use virtual rather than actual paths. For more on virtual paths, see "Directory Mappings", earlier in this chapter.

Language Extensions

The Web Listener can recognize various character sets, which are associated with various languages. The character set and language to be associated with a file is indicated with an extension to the filename. This is the character set and language to be used for communication between the Web Listener and web browsers. It is independent of the character set used for the data itself, which is indicated by the Web Agent parameter NLS_LANG.

This section defines the filename extensions that the Web Listener recognizes and the languages and character sets to which those extensions map. Extensions are case–sensitive. Each line in this section of the configuration file contains a language identifier as described in RFC 1766, followed by a character set, such as those described in RFC 1521, followed by one or more extensions that represent this combination of language and character set to the Web Listener. Multiple extensions are separated by blanks. For more information on *Oracle Web Listener's support for character sets, see "Oracle Web* Listener Language Extensions" in Chapter 3 of this manual.

An example follows:

Language Type	Character Set	File Extension(s)
en	iso–8859–1	eng
en	unicode–1–1	engUuc
fr–CA	iso–8859–1	frc
fr–FR	iso–8859–1	fr
jp–JP	iso–2022–jp	jp
jp–JP	unicode–1–1–utf–8	jpU

MAC003224

ORCL00941900

Since character sets are used only on textual files, language extensions differing only in the character set are equivalent for non–textual files. For example, for a Web Listener configured with the type extensions shown in the example, the file image.eng.jpg and the file image.uc.jpg both indicate an International English version of an image.

You can find more information and further examples under "Oracle Web Listener Authentication Configuration" in Chapter 3 of this manual.

MIME Types

The Multipurpose Internet Mail Extension protocol (MIME) defines a number of content types and subtypes that allow programs to recognize different kinds of files and deal with them appropriately. The MIME type specifies what kind of file it is, such as image, audio, or video, and the subtype gives the precise file format. For more information on MIME, you can look over the RFC documents that describe MIME at the following URL:

http://www.oac.uci.edu/indiv/ehood/MIME/MIME.html

This section defines MIME types known to the Web Listener by their file name extensions. Each line in this section names a MIME type and then lists extensions that map to that MIME type. Extensions are case–sensitive, and multiple extensions are separated by blanks.

Encoding Extensions

This section defines the Web Listener's known encoding types and their expected extensions. When the Web Listener encounters files with extensions defined here, it recognizes the file to be of the corresponding type. Each line in this section names an encoding type and the extensions that map to this type. Extensions are case–sensitive, and multiple extensions are separated by blanks.

Configure Security Form

Use this form to define the security protections for the Web Listener to use on various pages. The top of the form is shown in Figure 7 – 8. The rest of the form provides text fields in which to enter the security parameters and values of your choice.

MAC003225

ORCL00941901

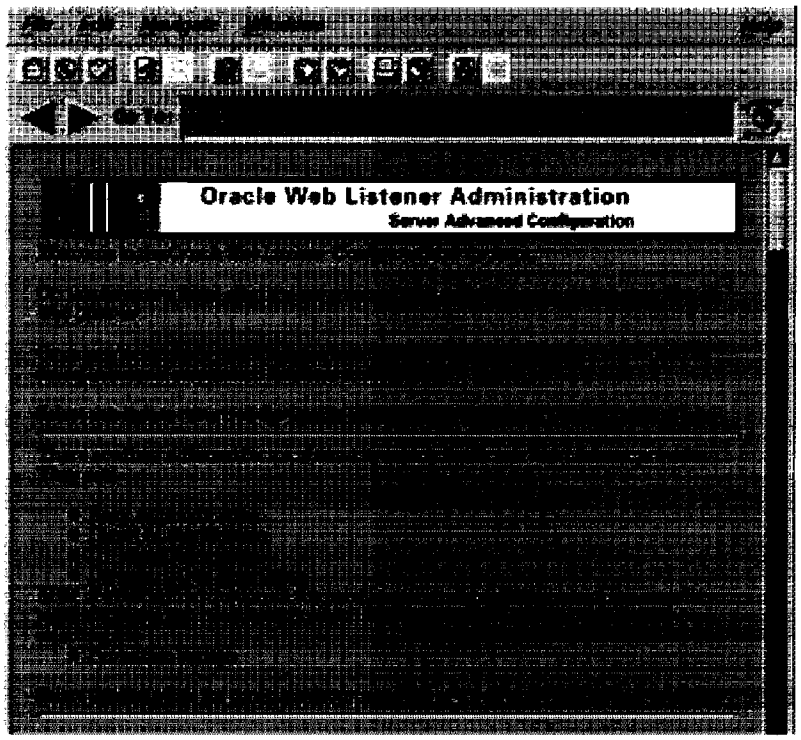


Figure 7 – 8 Web Listener Configure Security Form

There are two considerations:

- The security scheme. That is to say, how the Web Listener determines whether an access request comes from a sufficiently–privileged user. There are four possibilities, and you can impose more than one on the same material for extra security.
- Which users can access which files. You can define this both in terms of sets of users and sets of files. Specify files using virtual, rather than actual, paths. Virtual paths are explained under "Directory Mappings," earlier in this chapter.

The first of the following sections describes the security schemes in the abstract, the second describes the forms used to specify the users, and the third describes the form used to associate the users with the protected files or sets of files.

Click on a security scheme to navigate to the form which specifies the users in the manner relevant to that scheme, as described in the second section below. When you have completed these definitions, click

MAC003226

ORCL00941902

Protection Section to navigate to the form where you apply the security schemes by associating the specified users with the protected files. Alternatively, you can scroll down the Configure Security page to reach all these forms in sequence.

The Security Schemes

The four security schemes fall into two categories: authorization schemes and restriction schemes. Authorization schemes work on the principle of password protection—the user provides a password to verify his or her identity. Restriction schemes use the Internet itself to determine who the user is. Basic and Digest authentication are authorization schemes, while IP–based and Domain–based authentication are restriction schemes. All the schemes are explained in the paragraphs that follow.

Basic Authentication

Basic authentication employs usernames and passwords to restrict access to certain files on the Web Listener. Because Basic authentication sends passwords in the clear across the network, it is not recommended for extremely sensitive information.

Digest Authentication

Digest authentication works essentially the same way as Basic authentication, except that it does not send passwords in the clear. Instead, it uses a cryptographic checksum, often called a digest, to encode the password. This scheme is a significant improvement over Basic Authentication and should be used whenever possible. Unfortunately, this scheme depends on the cooperation of the browser to encrypt the password, and therefore does not work with older browsers that don't support this feature.

IP–based Restriction

IP–based restriction uses the network Internet Protocol (IP) address of the client to determine whether to allow or deny access to the protected page. Individual IP addresses or entire IP subnetworks may be specified. IP–based restriction is significantly more convenient than Basic and Digest authentication because it avoids the problems associated with passwords, such as keeping the password file secure and users forgetting their passwords. However, it is also less secure than Digest authentication because it is possible for a clever intruder to arrange to falsify his or her network address. Furthermore, many users need proxy servers because they are behind firewalls. In order to allow

MAC003227

ORCL00941903

access to such a user, the IP address of the proxy server needs to be specified. However, specifying the IP address of the proxy server provides access to the protected page to all users employing that proxy server.

Domain–based Restriction

Domain–based restriction is similar to IP–based restriction, except that it uses symbolic hostnames rather than network addresses. The use of symbolic hostnames is preferable because it is immune to changes in the underlying network architecture, which can result in changes to network addresses. Domain–based restriction, like IP–based restriction, is more convenient than Basic and Digest authentication, but also less secure than Digest authentication. As in IP–based restriction, it suffers from the problem with proxy servers.

Specifying The Privileged Users

After you have selected the security scheme or schemes to use, you need to specify the members of the privileged groups. The exact information that needs to be provided depends on the security scheme used.

Basic and Digest Authentication

For Basic and Digest Authentication, the information consists of a list of users, then groups, and then realms. Users and groups refer to the standard UNIX notion of users and groups. Realms are collections of groups that are assigned the same security privileges. In other words, they constitute a new (higher) level of the Unix grouping hierarchy. For example:

MAC003228

ORCL00941904

```
User Name    Password
-----------  -----------

mcarey:      achtung
mfoaf:       anything
chines:      pretend
pgabriel:    shockmonk

Group Name   User(s)
-----------  -----------

temps:       mcarey chines
sales:       chines mfoaf
analysts:    mfoaf pgabriel

Realms       Group(s)
-----------  -----------

red:         temps
blue:        sales temps
green:       analysts
```

In the above example, chines, whose password is pretend, belongs to the sales group, which is part of the blue realm. Thus, any files that are set to be accessible to the blue realm are accessible to chines as long as he types in pretend when prompted for his password. When specifying multiple users or groups, please make sure that they are separated by at least one space. Realms are non–exclusive, which is to say, users and groups can belong to multiple realms. A given realm can contain a mixture of users and groups.

IP–based Restriction

For IP–based restriction, you define groups of IP addresses that do or do not have access to a set of documents. For example:

```
IP Group     IP Address(es)
-----------  -----------

spyonly      +192.246.238.*  +204.4.214.*
notspy       –192.246.238.*  –204.4.214.*  +*
friends      +10.0.0.11  +10.0.0.83  +10.0.0.117
```

A '+' before an IP address denotes that that particular IP address or subnet should be given access. A '–' before an IP address denotes that that particular IP address or subnet should be denied access. An asterisk, as usual, is a wildcard indicating that the entire subnet—that is, all IP addresses that begin with the preceding numbers—is included. You can later use the group names given here to refer to the entire complex of associated IP addresses, with their permissions or lack of same, and assign file permissions to these groups.

MAC003229

ORCL00941905

In the above example, documents in the spyonly group are accessible only from sytems within the two specified subnets. Documents in not spy are accessible only from systems outside the two specified subnets. Documents in friends are accessible only from the three specified systems.

Domain–based Restriction

Domain–based restriction is similar to IP–based restriction. A list of groups and hostnames are specified listing which hosts should or should not have access to a set of documents. The following is an example of domain–based restriction:

```
Domain Group        Domain Name(s)
____________        ______________

oracle:             +*.oracle.com
misc:               +dilbert.redbull.com –*.redbull.com
                    –liz.lotion.com +*
```

You later can use the group names given here to refer to the entire complex of associated domain names, just as in IP–based restriction.

In the above example, documents in the group oracle are accessible only to systems within the oracle domain. Documents in misc are accessible to connections from the specific system dilbert.redbull.com, but not any other system in the redbull.com domain. Documents in misc are also accessible to anyone outside of the redbull.com domain as long as they are not connecting from liz.lotion.com

Performing the Security Assignments – The Protection Section

Each line in the Protection Section of the form contains a file specification followed by one or two security schemes to protect that group of one or more files.

You can specify files in any of the following three ways:

- by giving an explicit filename
- by giving an incomplete filename that ends in an asterisk to indicate that all files matching the portion before the asterisk are affected
- by giving a directory, in which case all files in that directory and all of its subdirectories are affected

Fill in the file specification according to these criteria and use the pop–up menus to apply the security schemes. All files are specified

MAC003230

ORCL00941906

using virtual, rather than actual, paths. Virtual paths are explained under "Directory Mappings," earlier in this chapter.

You can apply one authentication and/or one restriction scheme for each file specification. If you choose to use both, you must choose '&' or '|' from the '&|' pop–up menu to decide whether the user must satisfy both ('&') or only must satisfy either ('|') of the two schemes. Restriction schemes, naturally, are applied without user input.

To specify an authentication scheme, choose Basic or Digest from the pop–up menu. Then specify in the realm column the realm to which the scheme is to apply.

To specify a restriction scheme, choose IP or Domain from the pop–up menu. Then specify in the group column the group of IP addresses or Domain names, as appropriate, to which the scheme is to apply.

Here is an example:

Files	Basic/Digest	Realm	&/|	IP/Domain	Group
/secret/	Basic	red	|	IP	spyonly
/misc/hot*	Basic	green	&	IP	spyonly
/company/*				IP	spyonly
/update.htm	Basic	blue			

In this example, all of the files in /supersecret/, but not its subdirectories, can be accessed by any user who is either coming from the correct IP address or has a username and password for the red realm. In the second case, all files in /misc/ that start with "hot" can be accessed only by a user who has both the correct IP address and authentication for the green realm. Files in /company/, including subdirectories (as indicated by the asterisk (*)), are protected only by IP address. The single file update.htm is protected only by authorization in the blue realm.

MAC003231

ORCL00941907

MAC003232

ORCL00941908

CHAPTER

7

The Oracle WebServer Administration Utility

This chapter shows you how to use the Administration Utility of the Oracle WebServer. The Administration Utility enables you to configure the components of the Oracle WebServer from any web browser, so you do not have to manually edit files to perform common operations.

To navigate to the Administration Utility, click Administration from the Oracle WebServer home page. The utility is divided into the following three areas:

- The Oracle7 Database Server. This enables you to start up or shut down an Oracle7 database.
- The Oracle Web Listener. This enables you to create an instance of the Oracle Web Listener or to modify the parameters of an existing instance.
- The Oracle Web Agent. This enables you to create or modify web agent services, including the specification of password–protected security.

The Administration page is shown in Figure 7 – 1. Note: If you are viewing this document online, there are no screen shots provided.

MAC003233

ORCL00941909

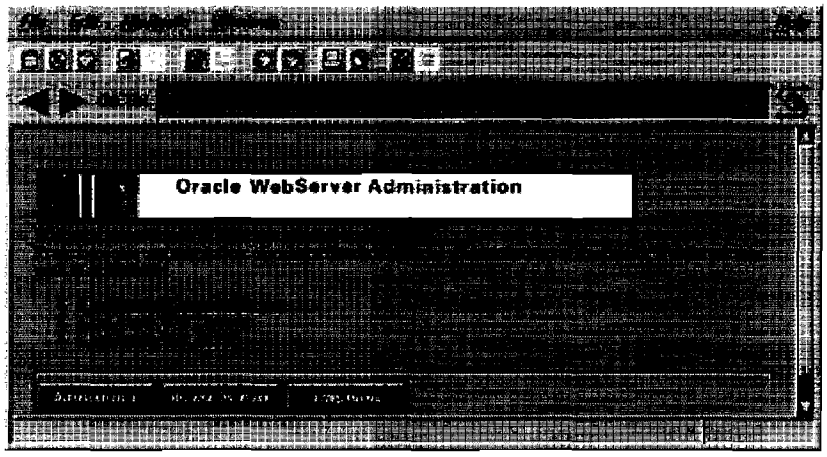


Figure 7 – 1 Administration Utility Form

Choose the area you want to use by clicking the appropriate button. This chapter contains a section for each administration area.

Database Administration

Use this page, shown in Figure 7 – 2, to start up or shut down any Oracle7 database running on the same machine as the Oracle Web Listener.



Figure 7 – 2 Database Administration Page

MAC003234

ORCL00941910

The page lists the available databases. This list comes from a file on the machine on which the Web Listener resides. On UNIX machines, this is known as the ORATAB file. See your platform–specific documentation for the location and format of this file. If no databases are listed, then the file contains no database descriptions.

The procedure is as follows:

1. Choose the database to which you want to apply the operation by clicking the radio button beside its name in the list.
2. Choose the operation you wish to perform and the mode by clicking the appropriate button. The modes are explained in the following sections.

For more information on database startup and shutdown, see the Oracle7 Server Administrator's Guide.

Startup Modes

You can perform database startup in either of the following two modes:

Normal

Starts up the database normally. This assumes that the database currently is shut down. If there are any errors during startup, they are reported, and the database is left shut down.

Force

Shuts down the database in the Abort mode (as explained under Shutdown Modes below), and then starts it up in the Normal mode.

Shutdown Modes

You can perform database shutdown in any of the following three modes:

Normal

Shuts down the database cleanly, waiting for current session to disconnect normally. New connections are not allowed while shutdown is in progress.

MAC003235

ORCL00941911

Immediate

Attempts to shut the database down immediately and cleanly. Disconnect any existing user sessions, but wait for active calls to complete before disconnecting. Roll back all uncommitted transactions .

Abort

Shuts the database down as quickly as possible, killing all active Oracle7 processes and freeing the memory used. This leaves current transactions unresolved. When you start up the database again, instance recovery is performed automatically. This option is recommended only for emergency use.

For more information on Oracle7 databases including startup and shutdown, see the Oracle7 Server Concepts Manual and the Oracle7 Server Administrator's Guide.

Web Agent Administration

The Web Agent Administration page enables you to create, delete, or modify Web Agent Services, which Web Agents use to connect to the database. An Oracle7 Web Agent Service specifies values needed by the Web Agent every time it makes a database connection. For example, an Oracle7 username and password are needed, as well as an Oracle7 SID or SQL*Net V2 service name. For more information on Web Agent Services, see "Web Agent Service" in Chapter 5.

Creating or Modifying Web Agent Services

Use the Web Agent section of the Administration Utility to create a new Web Agent Service or to copy, delete, or modify an existing one. The initial Web Agent Administratin page is shown in Figure 7 – 3.

MAC003236

ORCL00941912



Figure 7 – 3 Initial Web Agent Administration Page

To create a new Oracle Web Agent Service, click Create New Service. The Oracle Web Agent Administration Creation form appears.

To modify, copy, or delete an existing Oracle Web Agent Service, click the name of that service in the list provided. The Oracle Web Agent Administration Modification form appears.

The Oracle Web Agent Administration Service Creation Form

Use this fill–out form, shown in Figure 7 – 4 to enter the parameters for the service you want to create. The parameters are explained below.

MAC003237

ORCL00941913

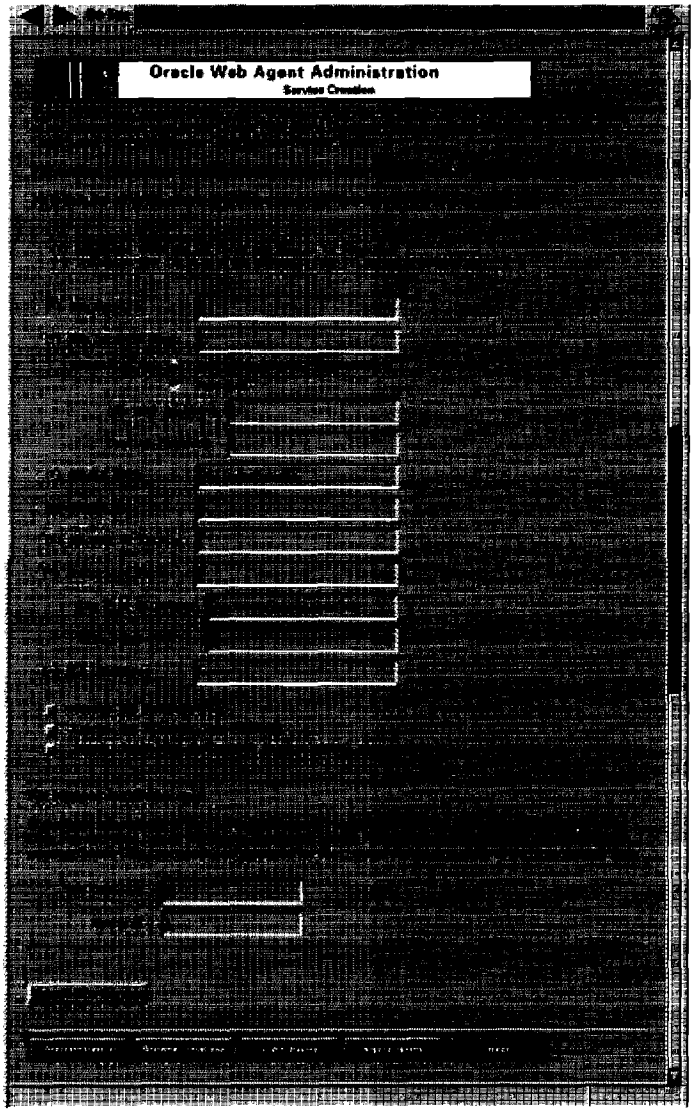


Figure 7 – 4 Create Web Agent Service Form

OWA Service:

The name of the service you want to create. The name is not case sensitive and must be unique.

MAC003238

ORCL00941914

OWA Database User:

The name of the Oracle7 user that will be used to access the database for this service. If operating system (OS) authentication is used, then the Web Listener must be run with the same effective user as the Oracle7 user. For more information on Oracle7 users, see "CREATE USER" in the Oracle7 Server SQL Reference.

Identified By

A radio button that determines whether Oracle7 or the Operating System performs authentication.

OWA User Password:

The Oracle7 password for the associated OWA Database User. If OS authentication is be used for the OWA Database User, as determined by the Identified By radio button, you can leave this field blank. Otherwise, the username and password for the OWA Database User are verified by Oracle7 upon submission of a Web Agent service.

If an error occurs when you attempt to submit this form, the password fields are blanked out and must be filled in again. This is to provide the tightest security currently available for your passwords. As a short–cut, you can select the Back button on your Web Browser to return to, and correct, the original submission form.

Confirm Password:

This field is used to verify the OWA user password you entered above before recording it.

ORACLE_HOME:

The location of the Oracle7 code tree in the file system. This should be the ORACLE_HOME for the database to which this Web Agent service connects, unless the Web Agent service is set up to connect to a remote database (over SQL*Net). In that case, specify the ORACLE_HOME where the Web Agent is installed.

ORACLE_SID:

The system ID of the database. You only need this if the Web Agent is to connect to a local database. For more on Oracle7 SIDs, see the Oracle7 Server Administrator's Guide.

SQL*Net V2 Service:

The SQL*Net service name that indicates which database to connect to. If the database is on the same machine as the Web Listener, then the SQL*Net V2 Service field can be left empty, and an Oracle7 SID specified instead. If both parameters are specified, the SQL*Net V2 Service takes precedence. If SQL*Net V1 is installed on this machine, then a SQL*Net V1 Connect String may also be specified here. For more on the SQL*Net V2 service name, see Understanding SQL*Net.

Authorized Ports:

To provide security for your Web Agent service, use the Authorized Ports field to list (space–delimited) the valid Web Listener network ports for the Web Agent to service. This field is required, and the network ports must be listed explicitly.

Log File Directory:

The directory where the Oracle Web Agent writes its error file. The error file will have the name service_name.err.

If the Web Agent is unable to resolve a service name, and no service called OWA_DEFAULT_SERVICE exists, then it will write error and logging information to the file owa.err.

The default value for the Log File Directory is the log directory under the OWS directory within ORACLE_HOME.

Note that these log files are different from those used by the Oracle7 Server for database recovery. The latter are explained in the Oracle7 Server Concepts Manual.

HTML Error Page :

If the Oracle Web Agent is unable to process a request, it returns this HTML page to the user making the request, and at the same time writes detailed errors and debugging information to the error message file in the Log File Directory. If this field is left blank, a default error message is returned to the client.

NLS Language :

This field specifies the NLS_LANG parameter, which specifies the character set to be used. If this field is left blank, then the Web Agent administration program looks it up in the configuration file for the

database upon service submission.

Create OWA Database User :

Create an Oracle7 user using the OWA user name and OWA user password provided. If the Identified by ... Operating System radio button is selected, then the user is created as identified externally, which means that the operating system, rather than Oracle, performs user authentication. In that case, this value is ignored.

Change OWA Database User Password:

Change the Oracle7 Database user's existing password to that specified by the OWA User Password. If the Identified by ... Operating System radio button is selected, then the user will be altered to be identified externally, which means that the operating system, rather than Oracle, performs user authentication.

Install Oracle WebServer Developer's Toolkit PL/SQL Packages

Create the Oracle WebServer Developer's Toolkit PL/ SQL packages—OWA, HTP, HTF, and OWA_UTIL—under the schema of the OWA Database User.

After filling in the above fields, click the Submit New Service button to create the service. To activate your new service, go to the Oracle WebServer Administration form and click Configure for the Web Listener you want to associate with this agent. Then go to "Directory Mappings" to create a virtual path, with the service name as the last part of the path, to the OWS/ bin directory. Virtual paths are discussed under "Directory Mappings", later in this chapter.

The Oracle Web Agent Administration Service Modification Form

This form contains the same fields as the Oracle Web Agent Administration Creation Form. The fields show their current values except for the password fields, which are left blank for security reasons. You can leave these fields blank unless you want to change the password.

MAC003241

ORCL00941917

At the bottom of the form are the following three buttons:

Copy Service

This creates a copy of the service, leaving the original intact. To do this, simply change the service name (the OWA service field) and any other parameters you wish to alter for the new copy and click this button. If you specify no new password, the original password is copied. After clicking this button, you have two services.

Modify Service

This changes the existing service. You can change the name or leave it intact as desired. Make the desired changes and click the button. After clicking this button, you have one service.

Delete Service

This destroys the service. For this purpose, it does not matter whether you change any parameters. After clicking this button, you have no service. Calls to the deleted service in existing Web pages or PL/ SQL blocks do not produce errors, however. Instead, they are reinterpreted as calls to OWA_DEFAULT_SERVICE.

Securing a Web Agent Service

The mechanism of "services" provides a security framework such that individual Web Agent services can be password protected by the Web Listener. For example, to password protect the hr service, you would do the following:

1. Create a virtual path for the hr service by specifying a Web Listener directory mapping. This mapping must use a value of 'C' to indicate an executable. For more information on setting the directory mappings, see Directory Mappings later in this chapter.
2. Create a realm consisting of those users and groups (UNIX only) who are to have access to the service. For instruction on how to do this, see Specifying the Privileged Users, later in this chapter.
3. Assign the realm to the service and specify either Basic or Digest Authentication. To learn how to perform the assignment, see Performing the Security Assignments, later in this chapter. To find out about Basic and Digest Authentication, see Security Schemes, later in this chapter.

MAC003242

ORCL00941918

For more information of Web Listener security in general, see "Security Definitions", later in this chapter, and "Client to Listener Security" in Chapter 3.

Web Listener Administration

The Oracle Web Listener is a reliable, high–performance HTTP server suitable for high–traffic web applications where quick response is crucial.

From the Web Listener Administration page, shown in Figure 7 – 5, you can move to other pages that provide a convenient way for you to do the following:

- Create a new Web Listener
- Change or copy an existing Web Listener
- Start a Web Listener
- Stop a Web Listener
- See whether an Oracle Web Listener is currently running.

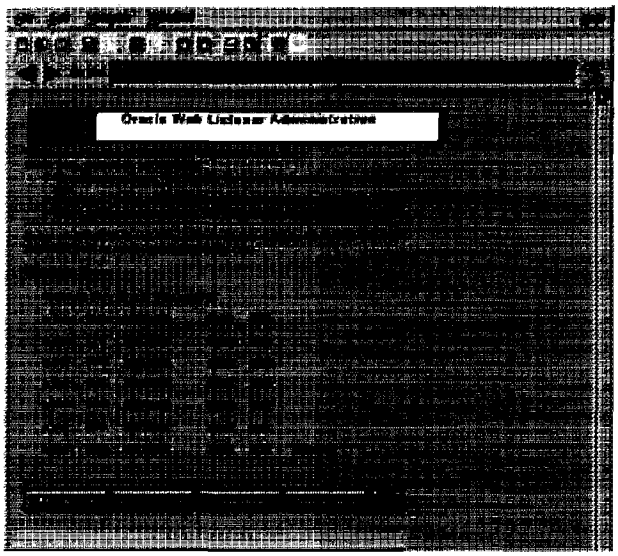


Figure 7 – 5 Web Listener Administration Page

To create a new Oracle Web Listener, click Create New Oracle Web Listener. This takes you to the Listener Creation form.

MAC003243

ORCL00941919

To perform any of the other operations, choose the appropriate existing Web Listener from the list provided. This list uniquely identifies each Web Listener by the network port on which it is configured to listen. For your convenience, the directory in which the configuration file resides is specified beside the port number. Unlike the port number, this directory need not be unique.

If a Web Listener is running, the word "running" appears beside it in the list.

To modify an existing Oracle Web Listener, select it in the list and click Configure. This takes you to the Configure Extended Parameters form, with the current parameter values filled in. Effect the changes by writing over the current values. For information on the parameters themselves, see "Configure Extended Parameters," later in this chapter.

To start a currently–stopped Oracle Web Listener, select it in the list and click Start.

To stop a currently–running Oracle Web Listener, select it in the list and click Stop.

> Note: You might need to perform a manual startup of an existing Web Listener if you inadvertently shut down the Web Listener that the Administration Utility itself is using. To find out how to do this, see "The WLCTL" in Chapter 3.

The Oracle Web Listener Creation Form

Use this form, shown in the next two figures, to create a new Oracle Web Listener by filling in the listed parameters. These are the parameters the Web Listener requires in order to run. After filling in all required parameters, press the appropriate button to choose a Basic Configuration or an Advanced Configuration. In either case, the Web Listener is created. If you choose Basic Configuration, the parameters you fill in here are all that are needed, and you can start the Web Listener you have created. If you choose Advanced Configuration, you go to the Configure Extended Parameters form to provide optional information. This is equivalent to selecting an existing Web Listener from the list in the initial Web Listener Administration page and clicking Configure. For more information on the Configure Extended Parameters form, see "The Configure Extended Parameters Form" later in this chapter.

MAC003244

ORCL00941920

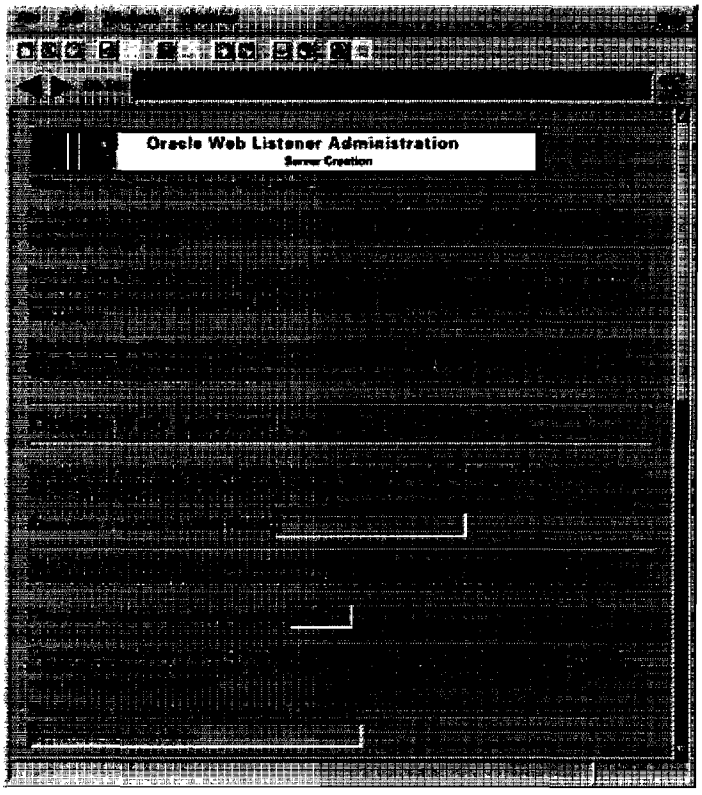


Figure 7 – 6 Listener Creation Form, Part 1

The Basic Configuration Parameters

The parameters required for a basic configuration are explained below:

Host Name

This is the name of the machine on which the Oracle Web Listener is to reside. It can either be an actual machine name or an alias in the form of a domain name, such as:

www. oracl e. com

Port Number

The Oracle Web Listener listens for network messages on a particular port. Use this parameter to specify that port. Numbers can range from 0 to 65535 with numbers below 1024 requiring superuser privileges on UNIX systems.

MAC003245

ORCL00941921

Configuration Directory

This is the directory where the configuration file for the Web Listener is to be located. A default is provided.

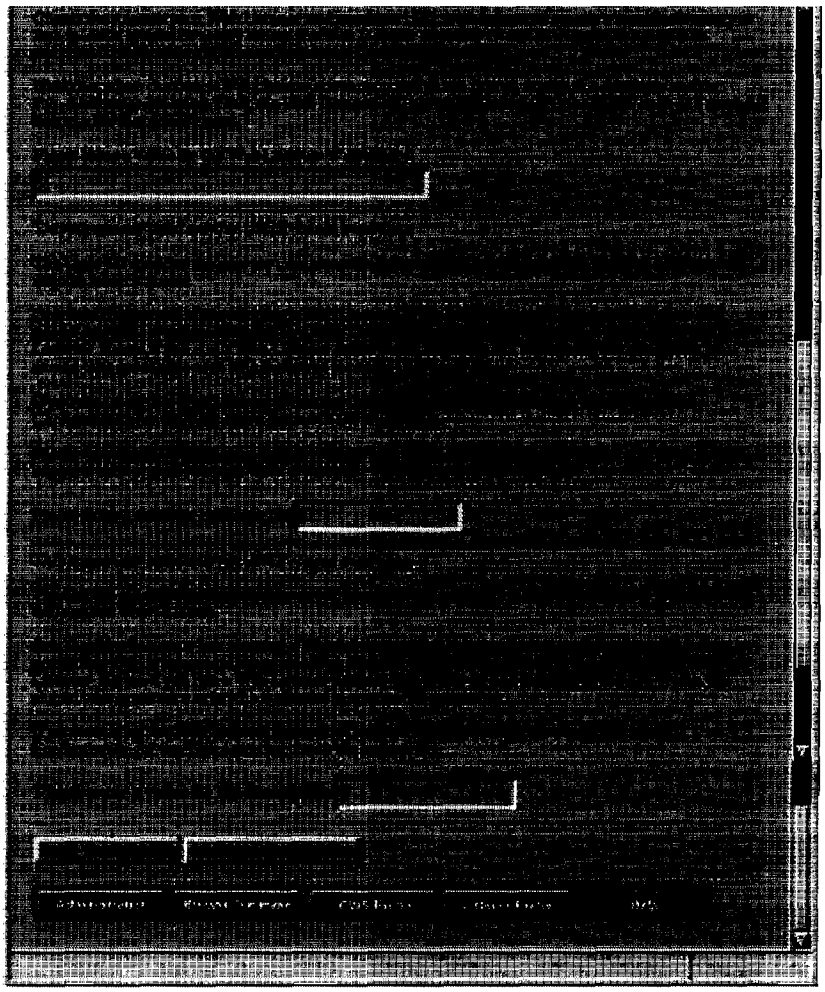


Figure 7 – 7 Listener Creation Form, Part 2

Web Listener Root (/) Directory

This is the directory that serves as the root (/) directory for the Web Listener. This is often called the "Document Root". It is used to resolve client calls that do not specify a directory. Thus a URL that does not contain a directory, such as http:/ / www.oracle.com/ index.html,

MAC003246

ORCL00941922

causes the Web Listener to retrieve the document "index.html" from the directory specified here.

User ID (UNIX only)

The User ID specifies the user (or user identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this user or by root with "Set user ID" set. The User ID can be either a username or a user ID number. The default is the effective ID. Specifying the Oracle owner or an Oracle user with the DBA role is not advised, as this compromises database security. For more information, see "roles" and "Database Administrators" in the Oracle7 Server Concepts Manual.

Group ID (UNIX only)

The Group ID specifies the group (or group identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this group or by root with "Set group ID" set. The Group ID can be either a group name or a GID number. The default is the effective GID.

Note that any Oracle Web Listener that invokes the Oracle Web Agent must have read access to the OWA configuration file (owa.cfg). However, since this file contains Oracle usernames and their corresponding passwords, it is recommended that read permissions only be given to the oracle owner and to one privileged group. This is the group that a Web Listener should run under if it is going to run the Oracle Web Agent.

A convenient group to use for this is the OracleDBA group. However, if you run an Oracle Web Listener under the DBA group, you will want to place strict controls on who can create CGI programs that will be invoked by that Web Listener. This is because the CGI program will run with group DBA privileges, and someone with malicious intent could potentially startup and shutdown your Oracle database, or even remove Oracle files. If you wish to avoid this potential security risk, then you need to do the following:

1. Create a new operating system group.
2. Make the Oracle owner a member of the group created in step 1.
3. Log in as the Oracle owner and change the group ownership of owa.cfg to the group created in step 1.

   Note: If at any future point in time you remove the owa.cfg file, you must reset its group ownership when it is recreated.

4. Configure the Group ID parameter of your Oracle Web Listener to be the name of the group created in step 1.
5. Restart your Administration Server (the Web Listener that the Administration Utility is using). You must do this to ensure that the Web Agent Administration program inherits the group created in step 1. Do not change the Group ID for the administration server. It must continue to run as the OracleDBA group.

Starting a New Oracle Web Listener

Once you have create a Web Listener, you are shown a Success! message. At this point, you are ready to start your new Oracle Web Listener. Ordinarily, you do this by navigating to the Oracle Web Listener Administration page and clicking start. Note that most web browsers buffer visited pages, and therefore do not reload them when you return to them using something like a Back button. For the new Web Listener to be listed as available for startup, you must navigate to it in such a way that the page is reloaded, for example by reentering the URL.

Reinitializing the Oracle Web Listener

If you selected Advanced Configuration and all the parameters you entered were valid, you will be shown a Success! message. You will then be presented with the Web Listener Configure Extended Parameters form which allows you to modify any of the Oracle Web Listener configuration parameters. It is important to remember that once the Web Listener is started, however, it does not automatically recognize new files added to its virtual file system or parameters changed in its configuration file. If you would like to add files to the Web Listener, you should navigate to the Web Listener Administration page and click Configure next to the Web Listener you want to modify.

The Configure Extended Parameters Form

This form provides a convenient way to customize an existing Oracle Web Listener for your particular environment. By using this form, you can avoid having to manually edit the Web Listener configuration file, thus eliminating the need to be familiar with the exact syntax for each parameter.

From this form, you will be able to customize the following:

- Network Parameters
- Log File Parameters

MAC003248

ORCL00941924

- Miscellaneous Web Listener Parameters
- User and Group Parameters
- Directory Mappings
- File Cache Definitions
- Language Extensions
- MIME Types
- Encoding Extensions
- Security Definitions for files and directories.

To go to the form for any of these sets of parameters, click it in the list. Alternatively, you can scroll through the Configure Extended Parameters Form to visit each parameter set in sequence except Security Definitions, which you can reach only by clicking.

These parameter sets are explained later in this chapter. For more information, see "Configuration Parameters" in Chapter 3.

Below each parameter set are three buttons. Click one to apply the changes you have made. The buttons are as follows:

Copy Web Listener

If you select this button, you must first change the Port Number listed in the Network Parameters section on this form. Then, the entire contents of this form will be used to create a new configuration file and thus a new Oracle Web Listener, leaving the existing Web Listener's configuration file intact. If you reached this form as part of a "create" operation, two Web Listeners are created.

Modify Web Listener

Pushing this button will overwrite the existing Oracle Web Listener's configuration file. This operation creates a new Web Listener and deletes the existing one. If you reached this form as part of a "create" operation, one Web Listener is created.

Delete Web Listener

Pushing this button will bring up a dialog for you to confirm the deletion. Confirming will do the following:

- Shut down the specified Oracle Web Listener if is currently running.

MAC003249

ORCL00941925

- Delete the specified Oracle Web Listener's configuration entry.
- Delete the specified Oracle Web Listener's configuration file.
- If you reached this form as part of a "create" operation, abort the create.

Network Parameters

This section defines the following parameters:

Host Name

The Host Name is the name of the machine on which the Web Listener executes. You must provide this parameter if the Web Listener cannot determine the host name or if the Web Listener executes under an alias. The Web Listener needs this information for logging purposes and to answer SNMP (Simple Network Management Protocol, an Internet standard protocol for checking the status of network components) requests for status. If no host name is specified here, the Web Listener uses the primary Internet address as the default.

Host Address

The Host Address is the IP address on which the Web Listener accepts connections. Specifying this address enables a Web Listener residing on a multi–homed machine to designate explicitly the direction from which it receives connections. Multi–homed machines are machines that are configured to have more than one IP address. Specifying ANY, which is the default, indicates that the Web Listener is willing to accept connections arriving on any of its available IP addresses. Leave the parameter set to ANY unless you have a specific reason for changing it.

Port Number

The Port Number is the advertised port on which the Web Listener listens for connections. Except for the value Port Number = 0, any port number is valid provided the Web Listener has the authority to use the requested port. Port numbers between 0 and 1023 are reserved ports and require root (or special) privileges in order to be used. Non–privileged processes can use any port greater than or equal to 1024 unless those ports are reserved for some other use. Ordinarily, the Web Listener uses 80 as the default port if no port number is specified

MAC003250

ORCL00941926

Max Connect Count

The Maximum Connect Count is the maximum number of simultaneous connections to the Web Listener that the Web Listener will accept. After the maximum count is reached, the Web Listener ignores requests for new connections. The default value is 50. The maximum and optimal value is 338.

DNS Resolution

Domain Name Service Resolution indicates whether and when the Web Listener will resolve hostnames from IP addresses. If the value of this parameter is set to ALWAYS, then the Web Listener always resolves the name. If it is set to LAZY, then the Web Listener resolves the name only on demand. If it is set to LAZY_WITH_CGI, then the Web Listener also resolves the name on demand for CGI applications. If it is set to NEVER, then the Web Listener never resolves the name, even on demand. The default is NEVER.

Web Listener PID

The Web Listener PID file is a one–line file that contains the Web Listener's process identifier. There is no default. This information can be used from a UNIX shell command line to stop the Web Listener, determine its status, or reinitialize a Web Listener that is already running. Normally, however, you use the Administration Utility itself to do these things.

Log File Parameters

The following parameters are defined in this section:

Log File Directory

The Log File Directory determines the directory where error and information logs are kept. You need to make sure that this directory actually exists, because the Web Listener will not create one if it does not. The Web Listener needs write permission on this directory and on the files it contains. There is no default.

Info File Name

The Log Information File contains periodic and event–driven messages that the Web Listener generates to help WebServer Administrators monitor their web site. The Oracle Web Listener records connections in the industry–standard Common Log Format. This format includes the

MAC003251

ORCL00941927

first line of the client's request, which specifies the action, URL, and protocol version. It also includes such information as the client's host address, whether the request was successful, and how many bytes were transmitted. This information may be used later to analyze what types of accesses are performed by which users, and to check for errors.

Error File Name

The Log Error File contains event–driven error messages that the Web Listener generates as it encounters problems with clients, resource access, and various other internal problems. There is no default.

Miscellaneous Web Listener Parameters

The following parameters are defined in this section:

Configuration Directory

This is the directory where the configuration file for the Web Listener is to be located. The current value is shown. If you change this value to a new directory, the file containing the changes you currently are making to all Web Listener parameters will be written to this new directory.

Initial File

The Initial File specifies the file that the Web Listener looks for when it is sent a directory URL. For example, if the Initial file is welcome, a request for http:/ / www/ stuff causes the Web Listener to return the file / stuff/ welcome. The default is initial.

User Directory

The User Directory is applicable only to Web Listeners running under the UNIX Operating System. It specifies the subdirectory in a user's directory the Web Listener searches when it receives a URL that starts with / ~username/ . For example, if at www, this value is set to public_html, then a request for http:/ / www/ ~charles/ initial.htm causes the Web Listener to return the file ~charles/ public_html/ initial.htm. There is no default.

User Directory Initial File

The User Directory Initial File is a special case. For directories accessed with a URL with / ~username/ , the Oracle Web Listener will not use HTTP incomplete file negotiation and so the Web Listener needs a

MAC003252

ORCL00941928

complete filename in the URL. The default is initial.htm.

Default MIME Type

The default MIME (Multimedia Internet Mail Extension) Type is the MIME type for the Web Listener to use if a URL uses a file name extension to request a MIME type the Web Listener does not support. MIME types specify the kind of file, such as image, audio, or video, and the subtype gives the precise file format. The default for this parameter is application/ octet–stream, which means the Web Listener will regard any unsupported MIME type as a binary file. If you would like the Web Listener to regard unsupported MIME types as anything other than application/ octet–stream, you need to set this parameter.

Default Character Set

This is the character set that the Web Listener uses if it does not recognize the character set specified in the URL's file extension. For more information, see "Language Extensions", later in this chapter.

Preferred Language

This is the language that the Web Listener uses if the specified or defaulted character set can apply to multiple languages, and the exact language is not specified. For more information, see "Language Extensions," later in this chapter.

Image Map

The contents of this field denote the extension that the Web Listener expects image maps to have. Thus, if your image maps have .map as their extension, set this parameter to map.

An image map is a graphic that is specially configured to act as a selector for one or more hyperlinks, based on where the user clicks on the graphic as displayed on the browser. Oracle WebServer processes image maps internally, without creating a separate process to handle them. An example of an image map would be a map of the world that allows the user to bring up an HTML document on a specific country by clicking on the country on the map. The Oracle Web Listener supports the same image map file formats as are read by the NCSA server.

MAC003253

ORCL00941929

Secure Information Parameters

The following parameters are applicable only to Web Listeners running on the UNIX Operating System:

User ID

The User ID specifies the user (or user identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this user or by root with "Set user ID" set. The User ID can be either a user name or a userid number. The default is the effective ID.

Group ID

The Group ID specifies the group (or group identification number) under whose privileges the Web Listener executes. To acquire these privileges, the Web Listener must be owned by this group or by root with "Set group ID" set. The Group ID can be either a group name or a GID number. The default is the effective GID. For further information on the use of Group ID for security, see "Basic Configuration Parameters," earlier in this chapter.

Directory Mappings

The Directory Mappings control the mapping of directories from the file system into the Web Listener's virtual file system. The Oracle Web Listener supports the concept of a "virtual" or mapped filesystem. This feature enables the URL used to reference a set of files (directory) to be kept constant even if the files are moved. The translation of the network visible "virtual" path to an actual path is performed at runtime based on the mappings set in this section.

You specify one mapping per line, putting the actual path on the left, the virtual path on the right, and selecting from the pop–up menu one of the following:

- 'C', which indicates that the directory contains CGI or other scripts. Although the directory can contain multiple CGI scripts, it cannot be recursive. That is to say, none of the CGI scripts can be contained in subdirectories.
- 'R', which indicates that the subdirectories should be mapped recursively. That is to say, subdirectories of the current directory are included.

MAC003254

ORCL00941930

- 'N', which indicates that the subdirectories should not be mapped recursively. That is to say, subdirectories of the current directory are not included.

Although you can list parameters in most sections of the configuration file in any order, you must order the parameters in this section so that parent virtual directories are listed before the virtual subdirectories they contain. Hence, the first directory mapping must reference the virtual root directory. The following is an example for UNIX systems:

File-System Directory	R/N/C	Virtual Directory
/html	R	/
/prodinfo	N	/products
/final/xr25	R	/products/xr25
/misc/people	R	/employees
/misc/routines	C	/scripts

In the example above, /html is the virtual root directory. This means that any URL that does not contain a directory gets mapped to /html. Thus, the URL http://www/foo.html will cause the Web Listener to look for foo.html in the /html directory. Since this particular mapping is set to be recursive, any URL which contains a directory other than /products, /employees, and /scripts will get mapped to /html/ directory. Thus, the URL http://www/foo/bar.html will cause the Web Listener to look for bar.html in the /html/foo directory. Since /prodinfo is set to non-recursive, any URL that contains a subdirectory of /products causes an error. The exception would be a URL that contains /products/xr25, because the third line maps this virtual directory to /final/xr25. The last line is an example of a mapping to a directory that contains CGI and other scripts. Thus, the URL http://www/scripts/baz will cause the Web Listener to execute /misc/routines/baz.

> Note: If you change the directory mappings, you must reload the Web Listener to put the new mappings into effect. To do this, use the WLCTL utility, as described under "WLCTL" in Chapter 3 of this document.

File Cache Definitions

This section designates cached files. Files listed here are kept open for the life of the Web Listener. Files not listed here are not cached, and the Web Listener releases the resources required to access them once there are no outstanding client references to them—that is, once their reference counts drop to zero.

MAC003255

ORCL00941931

File caching decreases the access time required for commonly–used files, but at the cost of increasing the memory that the WebServer process uses. For this reason, you should not cache very large files unless absolutely necessary.

You can specify files for caching in any of the following three ways:

- by giving an explicit filename
- by giving an incomplete filename that ends in an asterisk to indicate that all files matching the portion before the asterisk are affected
- by giving a directory, in which case all files in that directory and all of its subdirectories are affected

File references must use virtual rather than actual paths. For more on virtual paths, see "Directory Mappings", earlier in this chapter.

Language Extensions

The Web Listener can recognize various character sets, which are associated with various languages. The character set and language to be associated with a file is indicated with an extension to the filename. This is the character set and language to be used for communication between the Web Listener and web browsers. It is independent of the character set used for the data itself, which is indicated by the Web Agent parameter NLS_LANG.

This section defines the filename extensions that the Web Listener recognizes and the languages and character sets to which those extensions map. Extensions are case–sensitive. Each line in this section of the configuration file contains a language identifier as described in RFC 1766, followed by a character set, such as those described in RFC 1521, followed by one or more extensions that represent this combination of language and character set to the Web Listener. Multiple extensions are separated by blanks. For more information on Oracle Web Listener's support for character sets, see "Oracle Web Listener Language Extensions" in Chapter 3 of this manual.

An example follows:

Language Type	Character Set	File Extension(s)
en	iso–8859–1	eng
en	unicode–1–1	engUuc
fr–CA	iso–8859–1	frc
fr–FR	iso–8859–1	fr
jp–JP	iso–2022–jp	jp
jp–JP	unicode–1–1–utf–8	jpU

MAC003256

ORCL00941932

Since character sets are used only on textual files, language extensions differing only in the character set are equivalent for non–textual files. For example, for a Web Listener configured with the type extensions shown in the example, the file image.eng.jpg and the file image.uc.jpg both indicate an International English version of an image.

You can find more information and further examples under "Oracle Web Listener Authentication Configuration" in Chapter 3 of this manual.

MIME Types

The Multipurpose Internet Mail Extension protocol (MIME) defines a number of content types and subtypes that allow programs to recognize different kinds of files and deal with them appropriately. The MIME type specifies what kind of file it is, such as image, audio, or video, and the subtype gives the precise file format. For more information on MIME, you can look over the RFC documents that describe MIME at the following URL:

http://www.oac.uci.edu/indiv/ehood/MIME/MIME.html

This section defines MIME types known to the Web Listener by their file name extensions. Each line in this section names a MIME type and then lists extensions that map to that MIME type. Extensions are case–sensitive, and multiple extensions are separated by blanks.

Encoding Extensions

This section defines the Web Listener's known encoding types and their expected extensions. When the Web Listener encounters files with extensions defined here, it recognizes the file to be of the corresponding type. Each line in this section names an encoding type and the extensions that map to this type. Extensions are case–sensitive, and multiple extensions are separated by blanks.

Configure Security Form

Use this form to define the security protections for the Web Listener to use on various pages. The top of the form is shown in Figure 7 – 8. The rest of the form provides text fields in which to enter the security parameters and values of your choice.

MAC003257

ORCL00941933

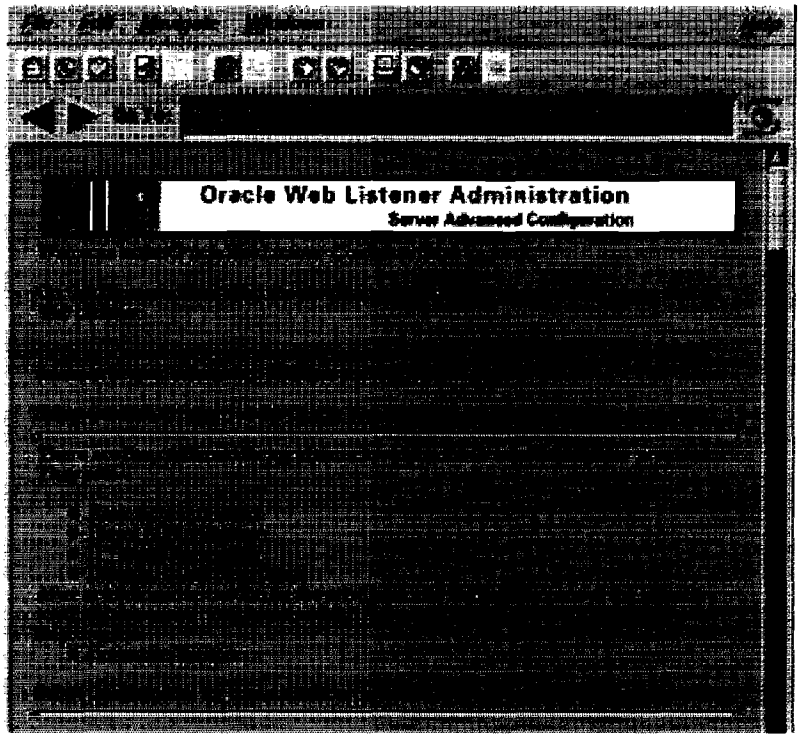


Figure 7 – 8 Web Listener Configure Security Form

There are two considerations:

- The security scheme. That is to say, how the Web Listener determines whether an access request comes from a sufficiently–privileged user. There are four possibilities, and you can impose more than one on the same material for extra security.
- Which users can access which files. You can define this both in terms of sets of users and sets of files. Specify files using virtual, rather than actual, paths. Virtual paths are explained under "Directory Mappings," earlier in this chapter.

The first of the following sections describes the security schemes in the abstract, the second describes the forms used to specify the users, and the third describes the form used to associate the users with the protected files or sets of files.

Click on a security scheme to navigate to the form which specifies the users in the manner relevant to that scheme, as described in the second section below. When you have completed these definitions, click

MAC003258

ORCL00941934

Protection Section to navigate to the form where you apply the security schemes by associating the specified users with the protected files. Alternatively, you can scroll down the Configure Security page to reach all these forms in sequence.

The Security Schemes

The four security schemes fall into two categories: authorization schemes and restriction schemes. Authorization schemes work on the principle of password protection—the user provides a password to verify his or her identity. Restriction schemes use the Internet itself to determine who the user is. Basic and Digest authentication are authorization schemes, while IP–based and Domain–based authentication are restriction schemes. All the schemes are explained in the paragraphs that follow.

Basic Authentication

Basic authentication employs usernames and passwords to restrict access to certain files on the Web Listener. Because Basic authentication sends passwords in the clear across the network, it is not recommended for extremely sensitive information.

Digest Authentication

Digest authentication works essentially the same way as Basic authentication, except that it does not send passwords in the clear. Instead, it uses a cryptographic checksum, often called a digest, to encode the password. This scheme is a significant improvement over Basic Authentication and should be used whenever possible. Unfortunately, this scheme depends on the cooperation of the browser to encrypt the password, and therefore does not work with older browsers that don't support this feature.

IP–based Restriction

IP–based restriction uses the network Internet Protocol (IP) address of the client to determine whether to allow or deny access to the protected page. Individual IP addresses or entire IP subnetworks may be specified. IP–based restriction is significantly more convenient than Basic and Digest authentication because it avoids the problems associated with passwords, such as keeping the password file secure and users forgetting their passwords. However, it is also less secure than Digest authentication because it is possible for a clever intruder to arrange to falsify his or her network address. Furthermore, many users need proxy servers because they are behind firewalls. In order to allow

MAC003259

ORCL00941935

access to such a user, the IP address of the proxy server needs to be specified. However, specifying the IP address of the proxy server provides access to the protected page to all users employing that proxy server.

Domain–based Restriction

Domain–based restriction is similar to IP–based restriction, except that it uses symbolic hostnames rather than network addresses. The use of symbolic hostnames is preferable because it is immune to changes in the underlying network architecture, which can result in changes to network addresses. Domain–based restriction, like IP–based restriction, is more convenient than Basic and Digest authentication, but also less secure than Digest authentication. As in IP–based restriction, it suffers from the problem with proxy servers.

Specifying The Privileged Users

After you have selected the security scheme or schemes to use, you need to specify the members of the privileged groups. The exact information that needs to be provided depends on the security scheme used.

Basic and Digest Authentication

For Basic and Digest Authentication, the information consists of a list of users, then groups, and then realms. Users and groups refer to the standard UNIX notion of users and groups. Realms are collections of groups that are assigned the same security privileges. In other words, they constitute a new (higher) level of the Unix grouping hierarchy. For example:

MAC003260

ORCL00941936

```
User Name    Password
---------    --------
mcarey:      achtung
mloaf:       anything
chines:      pretend
pgabriel:    shockmonk

Group Name   User(s)
----------   -------
temps:       mcarey chines
sales:       chines mloaf
analysts:    mloaf pgabriel

Realms       Group(s)
------       --------
red:         temps
blue:        sales temps
green:       analysts
```

In the above example, chines, whose password is pretend, belongs to the sales group, which is part of the blue realm. Thus, any files that are set to be accessible to the blue realm are accessible to chines as long as he types in pretend when prompted for his password. When specifying multiple users or groups, please make sure that they are separated by at least one space. Realms are non–exclusive, which is to say, users and groups can belong to multiple realms. A given realm can contain a mixture of users and groups.

IP–based Restriction

For IP–based restriction, you define groups of IP addresses that do or do not have access to a set of documents. For example:

```
IP Group     IP Address(es)
--------     --------------
spyonly      +192.246.238.*  +204.4.214.*
notspy       –192.246.238.*  –204.4.214.*  +*
friends      +10.0.0.11  +10.0.0.83  +10.0.0.117
```

A '+' before an IP address denotes that that particular IP address or subnet should be given access. A '–' before an IP address denotes that that particular IP address or subnet should be denied access. An asterisk, as usual, is a wildcard indicating that the entire subnet—that is, all IP addresses that begin with the preceding numbers—is included. You can later use the group names given here to refer to the entire complex of associated IP addresses, with their permissions or lack of same, and assign file permissions to these groups.

MAC003261

ORCL00941937

In the above example, documents in the spyonly group are accessible only from sytems within the two specified subnets. Documents in notspy are accessible only from systems outside the two specified subnets. Documents in friends are accessible only from the three specified systems.

Domain–based Restriction

Domain–based restriction is similar to IP–based restriction. A list of groups and hostnames are specified listing which hosts should or should not have access to a set of documents. The following is an example of domain–based restriction:

```
Domain Group        Domain Name(s)
------------        --------------
oracle:             +*.oracle.com
misc:               +dilbert.redbull.com -*.redbull.com
                    -liz.lotion.com +*
```

You later can use the group names given here to refer to the entire complex of associated domain names, just as in IP–based restriction.

In the above example, documents in the group oracle are accessible only to systems within the oracle domain. Documents in misc are accessible to connections from the specific system dilbert.redbull.com, but not any other system in the redbull.com domain. Documents in misc are also accessible to anyone outside of the redbull.com domain as long as they are not connecting from liz.lotion.com

Performing the Security Assignments – The Protection Section

Each line in the Protection Section of the form contains a file specification followed by one or two security schemes to protect that group of one or more files.

You can specify files in any of the following three ways:

- by giving an explicit filename
- by giving an incomplete filename that ends in an asterisk to indicate that all files matching the portion before the asterisk are affected
- by giving a directory, in which case all files in that directory and all of its subdirectories are affected

Fill in the file specification according to these criteria and use the pop–up menus to apply the security schemes. All files are specified

MAC003262

ORCL00941938

using virtual, rather than actual, paths. Virtual paths are explained under "Directory Mappings," earlier in this chapter.

You can apply one authentication and/or one restriction scheme for each file specification. If you choose to use both, you must choose '&' or '|' from the '&|' pop–up menu to decide whether the user must satisfy both ('&') or only must satisfy either ('|') of the two schemes. Restriction schemes, naturally, are applied without user input.

To specify an authentication scheme, choose Basic or Digest from the pop–up menu. Then specify in the realm column the realm to which the scheme is to apply.

To specify a restriction scheme, choose IP or Domain from the pop–up menu. Then specify in the group column the group of IP addresses or Domain names, as appropriate, to which the scheme is to apply.

Here is an example:

Files	Basic/Digest	Realm	&/|	IP/Domain	Group
/secret/	Basic	red	|	IP	spyonly
/misc/hot*	Basic	green	&	IP	spyonly
/company/*				IP	spyonly
/update.htm	Basic	blue			

In this example, all of the files in /supersecret/, but not its subdirectories, can be accessed by any user who is either coming from the correct IP address or has a username and password for the red realm. In the second case, all files in /misc/ that start with "hot" can be accessed only by a user who has both the correct IP address and authentication for the green realm. Files in /company/, including subdirectories (as indicated by the asterisk (*)), are protected only by IP address. The single file update.htm is protected only by authorization in the blue realm.

MAC003263

ORCL00941939

MAC003264

ORCL00941940

APPENDIX

A

Oracle WebServer Messages

This appendix contains messages you may receive if you encounter errors while using Oracle WebServer. If you get an error message, look in this appendix to find the probable cause and suggestions of what to do.

MAC003265

ORCL00941941

00001 – 00600 Generic Oracle WebServer Configuration Messages

OWS—00001 component: unable to open file name

Cause The file may not exist or may not be readable to the signalling program.

Action Verify the file's existence in the file–system. Check file and directory permissions and ownership, as well as the ownership of the signalling process.

OWS—00002 component: error reading file name

Cause Usually an operating system error. If the problem had been file permissions or non–existence, error 1 would have been signalled.

Action Check your operating system log for errors.

OWS—00003 component: error writing file name

Cause Usually an operating system error. If the problem had been file permissions or non–existence, error 1 would have been signalled.

Action Check your operating system log for errors.

OWS—00010 component: file format error when reading name

Cause The signalling program expected a specific file format which it did not find.

Action Analyze the file for incorrect formatting.

OWS—00011 Installation not completed. Root.sh must be run

Cause The signalling program checked if root.sh completed successfully.

Action Run root.sh as indicated by your installation guide.

OWS—00020 component out of memory when requesting number bytes

Cause The operating system was not able to supply the requested amount of memory.

Action Reduce the number of running programs on the machine and retry the operation.

OWS—00600 component internal error, arguments [arg1][arg2][arg3][arg4][arg5]

Cause This is the generic internal error number for Oracle WebServer program exceptions. This indicates that a process has encountered an exceptional condition.

Action Report as a bug. The first argument is the internal error number

MAC003266

ORCL00941942

05000 – 05499 Oracle Web Agent Messages

OWS—05100 Agent : unable to connect due to Oracle error number

Cause The Oracle Web Agent was unable to connect to the specified Database due to an Oracle error.

Action See the corresponding Oracle error in the Oracle7 Server Messages. Common errors are: invalid username/ password – check the Web Agent configuration file; Oracle not available – start up the database.

OWS—05101 Agent : execution failed due to Oracle error number

Action See the associated Oracle error.

OWS—05110 Agent : no stored procedure specified to call

Cause OWA requires the PATH_INFO environment variable to be set. PATH_INFO is set based on information passed through the URL after the OWA executable name and before the "?" or the end of the URL

Action Specify a stored procedure name after the OWA executable name in the URL.

OWS—05111 Agent : no procedure matches this call with the arguments passed

Cause The procedure and parameters passed to OWA don't match any existing stored procedures.

Action Verify that there is a stored procedure (not function)with no RECORD parameters which matches the parameters passed. If no value was passed for a parameter, be sure that the stored procedure was declared to have a default value for that parameter. Check for typographical errors.

OWS—05112 Agent : too many procedures matches this call

Cause The procedure and parameters passed to OWA match too many existing stored procedures.

Action This should only happen if there are overloaded stored procedures, and that in two different versions of the procedure, a given parameter is declared in one as a scalar and the other as a PL/ SQL table.

MAC003267

ORCL00941943

OWS—05150 Agent : Web Server Error – environment variable name is NULL or non–existent

Cause The Web Server or other invoking application did not set the listed environment variable.

Action Contact your Web Server or other invoking application vendor.

OWS—05151 Agent : Port number not in list of valid ports : number list

Cause The network port that the invoking an HTTP server is listening on is not a valid port for this OWA Service.

Action Add this port to the list of valid ports for this Service, or access this service through an HTTP server listening on a valid port.

OWS—05200 Agent : service name not found in configuration file

Cause The piece of the URL just before the Web Agent executable name did not correspond to a Web Agent Service in the OWA configuration file.

Action Correct the URL, or create the desired service.

OWS—05201 Agent : service parameter name not found for service name

Cause A parameter required by the Oracle Web Agent was not found in the configuration file along with the specified service.

Action Modify the Web Agent service configuration using the Web Agent administration form.

MAC003268

ORCL00941944

05500 – 05599 Oracle Web Agent Administration Messages

OWS—05500 Service name successfully written

OWS—05501 Service name successfully modified

OWS—05502 Service name successfully deleted

OWS—05503 Password for service name successfully modified

OWS—05504 Service name successfully written. Verify the PL/SQL output in filename

Cause The Web Agent Administration program successfully wrote the specified service and performed all actions requested. The program, however, cannot detect errors which may have occurred while installing the PL/ SQL.

Action Check the output in the file specified and verify that there are not unexpected error messages.

OWS—05520 Service name not deleted because it does not exist

OWS—05521 Service name submission failed because the passwords submitted do not match

Cause The password and the confirm password field do not match.

Action Retype both passwords and re–submit the form.

OWS—05522 Service name submission failed because password identification was selected and the submitted password was NULL

Cause The form submitted indicated that the Web Agent service should use a password to connect to the database, however no password was provided.

Action Select operating system authentication for this service, or provide a password for the given OWA Database User.

OWS—05523 Service name submission failed because value for mandatory parameter name is NULL

Action Fill in the specified field and re–submit the form.

MAC003269

ORCL00941945

OWS—05524 Service name submission failed because both parameter 1 and parameter2 are NULL

Cause Both of the listed fields were NULL.

Action Fill in one of the listed fields and re–submit the form.

OWS—05525 Service name submission failed because a service with the same name already exists

Action Choose another name for the service you are attempting to create, or delete the existing service.

OWS—05526 Service name submission failed due to error number

Action See the associated error and take the appropriate action.

OWS—05527 Service name submission failed because the usertype Username and Password are invalid

Cause Either the OWA Database User name and password, or the DBA username and password, is not a valid username/ password combination for the specified database.

Action Correct the username and password and re–submit the form.

OWS—05528 Service name submission failed because the DBA User submitted does not have privileges

Action Choose a different Oracle username and password for the DBA.

OWS—05529 Service name to modify does not exist

Cause The user chose to modify an existing Web Agent service, however that service does not exist. This error should not occur unless the service to modify was deleted from a different form after this Service Modification Form was generated.

Action Create a new service.

OWS—05530 Service name submission failed because the NLS Language could not be determined

Cause The Web Agent Administration package was unable to lookup the NLS Language for the specified database.

Action Specify the value explicitly or submit a valid dba username and password.

MAC003270

ORCL00941946

OWS—05531 Service name submission failed because neither Server Manager, nor SQL*DBA could be run

Cause The Web Agent Administration package was unable to install the WebServer Developer's Toolkit which must be done from Server Manager (line mode) or SQL*DBA.

Action Install either Server Manager or SQL*DBA in the ORACLE_HOME where the Web Agent administration is installed.

05600 – 05699 Oracle Web Database Administration Messages

OWS—05600 Startup of database name successful

OWS—05601 Shutdown of database name successful

OWS—05610 Startup of database name failed due to error number

Action See the associated error and take the appropriate action.

OWS—05611 Shutdown of database name failed due to error number

OWS—05620 DB Admin : no database selected

Action Return to the administration screen and select a database.

OWS—05621 DB Admin : name is already running.

Cause A request was made to start up a database, but that database is already running.

Action If you would like to restart the database, shut it down first or choose Startup Force.

OWS—05622 DB Admin : name is already shut down.

Cause A request was made to shut down a database, but that database is already shut down.

OWS—05623 DB Admin : parameter file name is larger than maximum size number bytes

Cause The initialization file listed was larger than the maximum allowable size.

Action Reduce the size of the initialization file.

MAC003271

ORCL00941947

05700 – 05799 Oracle Web Listener Configuration Messages

OWS—05700 Configuration for Web Listener on port number successfully written

OWS—05701 Configuration for Web Listener on port number successfully deleted

OWS—05710 Value submitted for parameter must not be NULL

Action Type a value for the specified parameter and re–submit the form.

OWS—05711 Port number is already in use by the Oracle Web Listener Administration Utility

Cause The port specified for Web Listener creation or modification is already in use by the Oracle WebServer Administration Utility.

Action Choose another port to run one of the Web Listeners on.

OWS—05712 An Oracle Web Listener is already configured to run on port number

Cause The port specified for service creation or modification is already in use by another Oracle Web Listener.

Action Choose another port to run one of the Web Listeners on.

OWS—05713 Value for parameter must be between minval and maxval

Action Specify a value between the two limits and resubmit the form.

OWS—05714 For ports less than number, the effective userid must be name

Cause Ports below that specified require superuser privileges to run a Web Listener on it.

Action Configure the server to run on a unrestricted port, or set the effective userid for this Web Listener the superuser.

MAC003272

ORCL00941948

OWS—05720 Web Listener on port number successfully started

OWS—05721 Web Listener on port number successfully stopped

OWS—05722 Web Listener on port number successfully stopped

OWS—05725 Web Listener on port number failed to start

Action See the associated error message.

OWS—05726 Web Listener on port number failed to reload

Action See the associated error message.

OWS—05727 Web Listener on port number was not stopped

Action See the associated error message.

05900 – 05999 Oracle Web Server Configuration Messages

OWS—05900 Web Listener on port number successfully stopped

OWS—05901 Please enter a value for field name.

Cause The specified field is a required field for registration.

Action Type a value into the specified field and re–submit the form.

MAC003273

ORCL00941949

Index

Symbols

[ ], (symbols explained), 4 – 35
& (symbol explained), 7 – 31
%ROWTYPE attribute, 4 – 19
%TYPE attribute, 4 – 15, 4 – 18
| (symbol explained), 4 – 35, 7 – 31

A

Administration Utility
  navigating to, 7 – 1
  structure of, 7 – 1
  to edit Web Listener configuration files, 3 – 11
  Web Listener security assignments using, 7 – 30
Advanced Configuration button, 7 – 12
aliases
  See also correlation variables; range variables; synonyms
  domain names for machine names, 7 – 13
  for database objects, 4 – 8
  for database tables, 4 – 10
  for paths, 7 – 22 to 7 – 27
ALTER TRIGGER statement (SQL), 4 – 37
anchors
  structure of, 2 – 13
  used in hypertext linking, 2 – 11
applications programming, PL/ SQL used for, 4 – 12 to 4 – 13
arrays, implemented as tables in PL/ SQL, 4 – 17
assignments
  initial, 4 – 15
  values to parameters (PL/ SQL), 4 – 20
  values to variables (PL/ SQL), 4 – 21
authorization schemes, 7 – 27 to 7 – 31
automatic recompilation (PL/ SQL), 4 – 31

B

Basic Authentication, 3 – 8, 3 – 17, 7 – 27, 7 – 28 to 7 – 31
Basic Configuration button, 7 – 12
blocks (PL/ SQL), 4 – 13
  See also subprograms (PL/ SQL)
Boolean logic
  IF statements and (PL/ SQL), 4 – 22
  Three–Valued in SQL, 4 – 7

C

cattributes, 6 – 5
cattributes parameter, use in passing exact text, 6 – 5
CGI, 3 – 4
CGI Application Execution, 3 – 5
CGI scripts, how flagged, 7 – 22 to 7 – 23
Change OWA Database User Password, 7 – 9

MAC003274

ORCL00941950

changes, viewing in HTML document, 2 – 16
character sets, 7 – 8 to 7 – 9
  default, 7 – 21
  international, defining, 7 – 24
columns
  See also tables; databases
  of database tables, 4 – 2
Common Gateway Interface. See CGI
compiler directives (PL/ SQL), 4 – 27
Configuration Directory, 7 – 14
  changing, 7 – 20
configuration files, Web Listener. See Web Listener, configuration file
Configure button, 7 – 12
Configure Extended Parameters Form, 7 – 16 to 7 – 31
Confirm Password, 7 – 7
connections, to the database, 4 – 4
constants, declaring (PL/ SQL), 4 – 16
constraints (SQL), 4 – 7
Copy Service button, 7 – 10
correlation variables
  in SQL statements, 4 – 10
  in triggers, 4 – 36
CREATE FUNCTION statement (SQL), 4 – 28
Create OWA Database User, 7 – 9
CREATE PACKAGE BODY statement (SQL), 4 – 32 to 4 – 33
CREATE PACKAGE statement (SQL), 4 – 31 to 4 – 32
CREATE PROCEDURE privilege (SQL), 4 – 29
  See also privileges; procedures
CREATE PROCEDURE statement (SQL), 4 – 28
CREATE TABLE statement (SQL), 4 – 7
cursor variables (PL/ SQL), 4 – 19
  See also cursors; variables
cursors (PL/ SQL), 4 – 19
  in package specifications, 4 – 31

D

Data Dictionary, 4 – 2
data structures, in PL/ SQL, 4 – 17 to 4 – 19
database links. See databases, distributed
databases
  adminstering from Administration Utility, 7 – 2
  changing data content of, 4 – 8 to 4 – 9
  connecting multiple, 4 – 5
  connecting to, 4 – 4, 7 – 4
  constraining, 4 – 7
  creating objects in using PL/ SQL, 4 – 13
  creating tables in, 4 – 7
  database triggers, 4 – 34 to 4 – 37
  distributed, 4 – 5
  interfacing to, 4 – 5
  keys used in, 4 – 2
  local, 4 – 5, 7 – 7
  missing data in, 4 – 7
  nulls used in, 4 – 7
  referencing objects in, 4 – 8
  relational, 4 – 2
  remote, 4 – 5, 7 – 7
  removing data from, 4 – 9
  retrieving data from, 4 – 6
  shutting down, 7 – 3
  specifying location of. See ORACLE_HOME
  starting up, 7 – 3
  storing code in, 4 – 13, 4 – 28 to 4 – 37
    See also stored procedures; packages; triggers
  structure of, 4 – 2 to 4 – 4
  unique identifiers in, 4 – 2
  users of, 4 – 4
datatypes
  composite, 4 – 14
  converting, 4 – 15
  inheritance of (PL/ SQL), 4 – 15
  PL/ SQL, 4 – 14
  pointer, 4 – 14
  RECORD (PL/ SQL), 4 – 18
  scalar, 4 – 14
  SQL, 4 – 7
    complex data, 4 – 4
    LONG RAW, 4 – 4
  structured, 4 – 14
  TABLE (PL/ SQL), 4 – 17
  user–defined, 4 – 16
    PL/ SQL records, 4 – 18 to 4 – 19
    PL/ SQL tables, 4 – 18

MAC003275

ORCL00941951

dblinks. See databases, distributed
DBMS. See RDBMS, databases
DBMS_SQL package, 4 – 13
DBMSSTDX.SQL script, 4 – 31
declarations
  constants (PL/ SQL), 4 – 16
  subprograms (PL/ SQL), 4 – 20 to 4 – 21
  types (PL/ SQL), 4 – 16
  variable (PL/ SQL), 4 – 15
DECLARE section (PL/ SQL), 4 – 14 to 4 – 21
Delete Service button, 7 – 10
DELETE statement (SQL), 4 – 9
Developer's Toolkit, 6 – 1
  installing, 7 – 9 to 7 – 10
  procedures and functions, 6 – 1 to 6 – 37
Digest Authentication, 3 – 8, 3 – 17, 7 – 27, 7 – 28 to 7 – 31
Directory Mappings, 7 – 22 to 7 – 23
distributed databases, 4 – 5
  See also databases
DNS Resolution, 7 – 19
Domain Name Based Restriction, 3 – 8
domain names
  as aliases for machine names, 7 – 13
  resolved to IP addresses, 7 – 19
  security based on. See Domain–based restriction
Domain–based restriction, 7 – 28, 7 – 30 to 7 – 31
dot notation
  for database objects, 4 – 8
  for PL/ SQL objects and subprograms, 4 – 16, 4 – 17

E

encoding, how indicated, 7 – 25 to 7 – 27
Error File Name, 7 – 20
error handling
  HTML, 7 – 8
  location of files, 7 – 19
  log file name, 7 – 20
  log files used for, 7 – 8
  Oracle7, 4 – 26
  PL/ SQL, 4 – 25 to 4 – 28
  return codes, 5 – 18
  system errors, 5 – 17 to 5 – 18
  triggers and, 4 – 36
exceptions (PL/ SQL), 4 – 25 to 4 – 28
  See also error handling
  declared in DECLARE section, 4 – 19
  declaring, 4 – 26
  predefined, 4 – 26
  user–defined, 4 – 27
EXECUTABLE section (PL/ SQL), 4 – 21 to 4 – 25
EXECUTE privilege (SQL), 4 – 29
  See also privileges
EXIT statement (PL/ SQL), 4 – 25

F

fields
  alternate name for database columns. See columns; databases
  in PL/ SQL records, 4 – 18
File Cache, 7 – 23 to 7 – 24
files
  caching of, 7 – 23
  encoded, 7 – 25 to 7 – 27
flow control (PL/ SQL), 4 – 21 to 4 – 25
FOR loops (PL/ SQL), 4 – 24
FOR UPDATE clause (PL/ SQL), 4 – 19
forced line breaks, 2 – 6
foreign keys, 4 – 2
  See also databases
  joins using, 4 – 10
Form tags, 2 – 22
  Input, 2 – 22
  TextArea, 2 – 22
Forms, 2 – 20
  selection menus, 2 – 24 to 2 – 26
  syntax, 2 – 21
FTP, 2 – 12
functions, 4 – 20 to 4 – 21
  See also subprograms (PL/ SQL)
  grouped into packages, 4 – 30
  return values of (PL/ SQL), 4 – 20 to 4 – 21

MAC003276

ORCL00941952

storing in database, 4 – 28 to 4 – 37

G

GOTO statement (PL/ SQL), 4 – 25
  error handling and, 4 – 28
graphics
  as hypertext buttons, 7 – 21 to 7 – 22
  within HTML documents, 2 – 14 to 2 – 16
Group ID, 7 – 15, 7 – 22

H

Host Address, 7 – 18
Host Name, 7 – 13, 7 – 18
HTF. See HyperText Functions
HTML
  basic concepts, 2 – 2
  case sensitivity, 2 – 3
  error handling, 7 – 8
  introduction, 2 – 1
  multi–button graphics, 7 – 21 to 7 – 22
HTML error page, 7 – 8
HTML Tags
  ADDRESS, 2 – 17
  BLOCKQUOTE, 2 – 7
  BODY, 2 – 4
  BOLD, 2 – 16
  CAPTION, 2 – 19
  definition list, 2 – 10
  escape sequence, 2 – 17
  getting started, 2 – 3 to 2 – 6
  HEAD, 2 – 4
  heading level, 2 – 5 to 2 – 10
  I, 2 – 16
  lists, 2 – 8
  mono spaced, 2 – 16
  nested list, 2 – 9
  ordered list, 2 – 8
  P, 2 – 5
  PRE, 2 – 5
  TABLE, 2 – 18
  TD, 2 – 18
  TH, 2 – 19
  TITLE, 2 – 4
  TR, 2 – 18
  U, 2 – 16
  unordered list, 2 – 9
HTP. See HyperText Procedures
HTTP, 2 – 12
HyperText Functions, description of, 5 – 2
HyperText Markup Language, 2 – 1
HyperText Procedures
  body tags, 6 – 11 to 6 – 16
  character format tags, 6 – 19
  definition of, 6 – 1
  description of, 5 – 2
  form tags, 6 – 23 to 6 – 30
  head–related tags, 6 – 8
  list tags, 6 – 15 to 6 – 20
  physical format tags, 6 – 22
  printing procedures, 6 – 5 to 6 – 7
  structural tags (HTML), 6 – 7 to 6 – 9
  table tags, 6 – 29 to 6 – 33

I

Identified by
  defined, 7 – 9
  radio button, 7 – 7
IF statement (PL/ SQL), 4 – 22
Image Map, 7 – 21 to 7 – 22
imagemap, 3 – 7
Info File Name, 7 – 19
Initial File, 7 – 20
initialization section (PL/ SQL packages), 4 – 33
input parameters. See parameters, modes of (PL/ SQL)
INSERT statement (SQL), 4 – 9
Internet Protocol based Restriction, 3 – 8
IP addresses
  Domain Name Resolution of, 7 – 19
  security based on. See IP–based restriction
  Web Listener, 7 – 18
IP–based restriction, 3 – 18, 7 – 27 to 7 – 28, 7 – 29 to 7 – 31
ISINDEX, 3 – 5, 3 – 6
ISMAP attribute, 2 – 14

MAC003277

ORCL00941953

J

joins, 4 – 10 to 4 – 11
  natural, 4 – 10
  outer, 4 – 10

L

labels, PL/ SQL, 4 – 25
languages
  See also character sets; SQL; PL/ SQL
  associating with character sets, 7 – 24 to 7 – 25
  default, 7 – 21
links
  to other documents, 2 – 11
  to other sections, 2 – 15
list tags
  definition, 2 – 10
  nested, 2 – 9
  ordered, 2 – 8
  types of, 2 – 8
  unordered, 2 – 9
Log File Directory, 7 – 8, 7 – 19
log files
  default, 7 – 8
  Web Agent Service, 7 – 8
loops (PL/ SQL), 4 – 23
  exiting nested, 4 – 25
  FOR, 4 – 24
  WHILE, 4 – 24

M

mailto, 2 – 12
Max Connect Count, 7 – 19
MIME (Multimedia Internet Mail Extension) types
  default, 7 – 21
  defining, 7 – 25
  further information, 7 – 25
Modify Service button, 7 – 10
multimedia data
  file types of, 7 – 25
  in Oracle7 database, 4 – 4
  MIME types, 7 – 21

N

naming conventions
  PL/ SQL, 4 – 17
    tables, 4 – 18
  SQL, 4 – 8
natural joins, 4 – 10
network communications, 4 – 5
  port selection, 7 – 8
NLS Language, 7 – 8 to 7 – 9
NOT NULL constraint (SQL), 4 – 7
  See also nulls; constraints
  PL/ SQL use of, 4 – 15
NULL statement (PL/ SQL), 4 – 23
nulls
  as Boolean values, 4 – 7
  database contains, 4 – 7
  differentiated from NULL statement, 4 – 23
  outer joins generate, 4 – 10
  prohibiting, 4 – 7, 4 – 15
  SQL handling of, 4 – 7
  UPDATE statement (SQL) generates, 4 – 9
  variables initialized to (PL/ SQL), 4 – 15

O

objects
  aliases for database objects. See correlation variables; synonyms
  ownership of database, 4 – 8
    See also databases
Oracle Web Agent, description of, 5 – 2 to 5 – 4
ORACLE_HOME
  default log file directory in, 7 – 8
  Web Agent Service uses, 7 – 7
ORACLE_SID, 7 – 7
Oracle7 Server, 4 – 1
  See also databases
  stores multimedia data, 4 – 4
outer joins, 4 – 10
output parameters. See parameters, modes of (PL/ SQL)

MAC003278

ORCL00941954

owa, calling procedures with, 7 – 4
OWA Database User, 7 – 7
OWA Log Directory, defines Web Agent error log, 5 – 19
OWA Service, 7 – 6
OWA Service Name, defines Web Agent error log, 5 – 19
OWA User Passord, 7 – 7
OWA Utilities package, description of, 6 – 2
owa.cfg
  See also Web Listener configuration file
  sample with parameters, 5 – 6 to 5 – 8
  Web Agent uses, 5 – 4
owa.err, 7 – 8
  See also error handling
OWA_DEFAULT_SERVICE, 7 – 8
  See also Web Agent Services
  deleted services and, 7 – 10
OWA_UTIL Package, 6 – 31 to 6 – 36
OWA_UTIL package, definition of, 5 – 2
OWAINS.SQL sql script, using to install Developer's Toolkit, 6 – 3

P

packages, 4 – 30 to 4 – 34
  bodies, 4 – 32
  creating, 4 – 31
  specification, 4 – 31
parameters
  actual (PL/ SQL), 4 – 20
  formal (PL/ SQL), 4 – 20
  in PL/ SQL, 4 – 20
  in PL/ SQL packages, 4 – 33
  log file, 7 – 19 to 7 – 20
  modes of (PL/ SQL), 4 – 20
  network, 7 – 18 to 7 – 19
  Web Agent Service, 7 – 5 to 7 – 9
parent keys, 4 – 3
  See also primary keys; foreign keys; databases
passwords
  changing, 7 – 9
  database, 4 – 4
  Web Agent Service uses, 7 – 7, 7 – 10 to 7 – 11
  Web Listener, 7 – 28
PID (Process Identifier), Web Listener, 7 – 19
PL/ SQL, 4 – 12 to 4 – 37
  blocks, 4 – 13
  declarations, 4 – 14 to 4 – 21
  DECLARE section, 4 – 14 to 4 – 21
  EXCEPTION section, 4 – 25 to 4 – 28
  EXECUTABLE section, 4 – 21 to 4 – 25
  flow control in, 4 – 21 to 4 – 25
  general discussion of, 4 – 12 to 4 – 13
  generating dynamically, 4 – 13
  GOTO restrictions, 4 – 25
  overloading subprograms, 4 – 33
  scope of objects and subprograms, 4 – 16
  sections of, 4 – 13
  subprograms, 4 – 13
  user–defined datatypes in, 4 – 16
Port Number, 7 – 13, 7 – 18
ports
  changing to copy Web Listeners, 7 – 17
  reserved, 7 – 18
  selecting, 7 – 8
  Web Listeners associated with, 7 – 13, 7 – 18
  Web Listeners identified by, 7 – 11
pragmas (PL/ SQL), 4 – 27
predicates
  in IF statements (PL/ SQL), 4 – 22
  in SQL, 4 – 6
    See also Three–Valued Logic
primary keys, 4 – 2
  enforcing, 4 – 7
private synonyms, 4 – 8
  See also synonyms
privileges
  See also roles; profiles; users
  object, 4 – 4
  required to create triggers, 4 – 36
  stored procedures require, 4 – 29
  stored procedures use, 4 – 29
  system, 4 – 4
procedures
  See also subprograms (PL/ SQL)
  declaring, 4 – 20 to 4 – 21
  grouped into packages, 4 – 30
  storing in database, 4 – 28 to 4 – 37
procedures and functions, getting started, 6 – 4

MAC003279

ORCL00941955

profiles, 4 – 5
  See also users, database
public synonyms, 4 – 8
  See also synonyms
  for WebServer paths. See virtual paths

Q

queries (SQL), 4 – 6
  multiple tables used in, 4 – 10 to 4 – 11
  stored in cursors (PL/ SQL), 4 – 19
  within other queries (subqueries), 4 – 11
query string, 3 – 5
QUERY_STRING environmental variable,
    passing parameters with, 5 – 10

R

RAISE statement (PL/ SQL), 4 – 27
  See also error handling
range variables, 4 – 10
  See also correlation variables
RDBMS, 4 – 1
  See also Databases
realms
  assigning, 7 – 31
  defined, 7 – 28 to 7 – 29
records
  alternate name for database rows. See rows;
      databases
  PL/ SQL datatype, 4 – 18
ref cursors (PL/ SQL), 4 – 14
  See also cursors
referenced keys, 4 – 3
referential integrity, 4 – 3
relational databases, 4 – 2
  See also databases
reload, 2 – 16
REQUEST_METHOD environmental variable,
    passing parameters with, 5 – 9
restriction schemes, 7 – 27 to 7 – 31
RETURN statement (PL/ SQL), 4 – 20 to 4 – 21
return values
  cursor (PL/ SQL packages), 4 – 31
  functions (PL/ SQL), 4 – 20 to 4 – 21
roles, 4 – 5
  See also privileges
rows
  See also tables; databases
  of database tables, 4 – 2

S

schemas, 4 – 8
  See also databases; users
scope
  of PL/ SQL objects and subprograms, 4 – 16
  PL/ SQL packages and, 4 – 33
SCRIPT_NAME variable, how Web agent uses,
    5 – 4
scripts, 3 – 5
security
  types of, 7 – 27 to 7 – 28
  Web Listener, 7 – 25 to 7 – 31
    assignments, 7 – 30 to 7 – 31
SELECT statement (SQL), 4 – 6
  See also queries (SQL)
servers
  database, 4 – 1
    See also databases; Oracle7 server
  web
    generic. See HTTP servers
    Oracle. See Web Listeners
service_name.err, Web Agent error log, 5 – 19
services
  SQL*Net V2, 7 – 8
  Web Agent. See Web Agent Services
shutdown, Web Listener, 7 – 12
shutdown mode (database), 7 – 3
SQL, 4 – 5 to 4 – 12
  building applications using, 4 – 12 to 4 – 13
  dynamic, 4 – 13
  predicates, 4 – 6
  standards, 4 – 5
  Three–Valued Logic (TRUE, FALSE, NULL)
      in, 4 – 7
SQL*Net, 4 – 5
  V2 service specfication, 7 – 8
SQLCODE, 4 – 26

MAC003280

ORCL00941956

SQLERRM, 4 – 26
STANDARD package (PL/ SQL), 4 – 26
Start button, 7 – 12
startup, Web Listener, 7 – 12
startup mode (database), 7 – 3
Stop button, 7 – 12
stored procedures, 4 – 28 to 4 – 34
  invoking in URL, 7 – 4
Structured Query Language. See SQL
Submit New Service button, 7 – 9
subprograms (PL/ SQL), 4 – 13
  declaring, 4 – 20 to 4 – 21
  error handling and, 4 – 27
  GOTO restrictions, 4 – 25
  overloading, 4 – 33
  parameters of, 4 – 17
  resolving ambiguous references, 4 – 17
  scope and visibility of, 4 – 16
subqueries (SQL), 4 – 11
subtypes (PL/ SQL), 4 – 16
  See also datatypes
  using to create PL/ SQL records, 4 – 18 to 4 – 19
  using to create PL/ SQL tables, 4 – 18
synonyms, for database objects, 4 – 8

T

Table attributes
  Align, 2 – 19
  Border, 2 – 19
  Colspan, 2 – 19
  Colspec, 2 – 20
  Nowrap, 2 – 19
  Rowspan, 2 – 20
  Valign, 2 – 19
Tables, in HTML, 2 – 18 to 2 – 20
tables
  database, 4 – 2, 4 – 4
    See also databases
    aliases for, 4 – 8
      See also range variables; synonyms
    creating, 4 – 7
    stored in cursors (PL/ SQL), 4 – 19
  HTML, 4 – 4
  ownership of, 4 – 8
  passing as parameters in PL/ SQL, 4 – 17
  PL/ SQL, referencing, 4 – 18
  PL/ SQL datatype, 4 – 17 to 4 – 18
text–only browsers, 2 – 14
Three–Valued Logic (TRUE, FALSE, NULL), 4 – 7
triggers, 4 – 34 to 4 – 37
  altering, 4 – 35 to 4 – 36
  creating, 4 – 35 to 4 – 36
  enabling and disabling, 4 – 37

U

Uniform Resource Locator (URL), 2 – 11, 2 – 13, 2 – 14, 3 – 5
unique identifiers
  See also primary keys
  database use of, 4 – 2
  in database. See naming conventions
unique keys, 4 – 3
UPDATE statement (SQL), 4 – 9
URL, 2 – 11
  See also Uniform Resource Locator (URL)
  Web Agent Service specified in, 7 – 4
URL encoding, 3 – 6
User Directory, 7 – 20
User Directory Initial File, 7 – 20
User ID, 7 – 15, 7 – 22
  Web Listener, 7 – 15, 7 – 22
users
  assigning to realms, 7 – 28
  assigning Web Listener security schemes to, 7 – 30
  authentication used for, 7 – 28
  database, 4 – 4
    controlling actions of, 4 – 4
    creating, 7 – 9
    own objects, 4 – 4
    own schemas, 4 – 8
    privileges and, 4 – 4
    privileges of, 7 – 8
      See also privileges
    use of resources, 4 – 5
    use roles, 4 – 5

MAC003281

ORCL00941957

directories used for, 7 – 20
domain name identification of, 7 – 30
IP identification of, 7 – 29 to 7 – 30
Oracle7, 4 – 4
OS authentication of, 7 – 7
Web Agent Services emulate, 7 – 7
Web Listener security assignments, 7 – 30 to 7 – 31

V

variables
assigning values to (PL/ SQL), 4 – 21
cursor, 4 – 14, 4 – 19
datatypes of. See datatypes
declaring (PL/ SQL), 4 – 15
in triggers, 4 – 36
initializing, in PL/ SQL packages, 4 – 33
instantiation in PL/ SQL packages, 4 – 31
virtual paths, 7 – 22 to 7 – 23
file caching uses, 7 – 24
security schemes use, 7 – 26
Web Agent Services use, 7 – 9
visibility
of PL/ SQL objects and subprograms, 4 – 16
PL/ SQL packages and, 4 – 33

W

Web Agent
CGI environment variables, 5 – 16
components of, 5 – 2
creating customized error pages, 5 – 17
Creation form (HTML), 5 – 6
error handling, 5 – 17 to 5 – 19
error log, 5 – 19
how it works, 5 – 3 to 5 – 7
optimizing multiple services, 6 – 3
overview, 5 – 1
request methods, 5 – 9
users of, 5 – 2
version number, 5 – 19
Web Agent error handling, application errors, 5 – 17
Web Agent Modification form, 7 – 9 to 7 – 10
Web Agent Service, 5 – 5 to 5 – 36
Web Agent Services
activating, 7 – 9
administering, 7 – 4 to 7 – 11
creating, 7 – 5, 7 – 9
deleting, 7 – 9
modifying, 7 – 5, 7 – 9
securing, 7 – 10 to 7 – 11
URL specifies, 7 – 4
Web Listener configuration file, 3 – 3
parameters in, 3 – 11 to 3 – 20
Web Listener Control Utility, 3 – 10 to 3 – 12
Web Listener Creation Form, 7 – 12
Web Listener PID, 7 – 19
Web Listener root directory, 7 – 14
Web Listeners
administering, 7 – 11 to 7 – 31
ascertaining state of, 7 – 12
associating with machines, 7 – 13, 7 – 18
associating with Web Agent Services, 7 – 9
configuration files, 7 – 16
configuration parameters for, 3 – 10
configuring, 7 – 16 to 7 – 31
connections to, maximum, 7 – 19
copying, 7 – 17
creating, 7 – 11
default directories, 7 – 20
deleting, 7 – 17 to 7 – 18
directory mapping, 3 – 3
directory mapping parameters, 3 – 13
DNS resolution, 3 – 7
encoding definitions, 3 – 14
error handling, 7 – 19 to 7 – 20
features, 3 – 2
file cache definitions, 3 – 13
file caching, 3 – 2, 7 – 23
group ID, 7 – 15, 7 – 22
HTTP protocol, 3 – 4
imagemap support, 3 – 7
international characters sets, 7 – 24 to 7 – 25
IP addresses, 7 – 18
language extensions, 3 – 3, 3 – 15
logging, 3 – 9
logging parameters, 3 – 12
memory mapping, 3 – 2
mime type definitions, 3 – 15
modifying, 7 – 12, 7 – 17
multimedia data, 7 – 25

MAC003282

ORCL00941958

overview, 3 – 1
parameters, 7 – 20 to 7 – 21
password protect Web Agent Services, 7 – 10
process identifiers (PIDs), 7 – 19
reinitializing, 7 – 16
reloading with WLCTL, 3 – 10
root directory of, 7 – 14 to 7 – 15
security, 3 – 8 to 3 – 10, 7 – 22, 7 – 25 to 7 – 31
  basic authentication, 7 – 28 to 7 – 31
  digest authentication, 7 – 28 to 7 – 31
  domain–based restriction, 7 – 30 to 7 – 31
  encoding, 7 – 25 to 7 – 27
  IP–based restriction, 7 – 29 to 7 – 31
  types of, 7 – 27 to 7 – 28
security for individual files, 3 – 18
  starting, 7 – 16
  starting with WLCTL, 3 – 10
  stopping with WLCTL, 3 – 10
  tracking information, 7 – 19
web servers
  generic. See HTTP servers
  Oracle. See Web Listeners
WebServer, administration, 7 – 19 to 7 – 20
WebServer Developer's Toolkit
  installing, 6 – 2
  security considerations, 6 – 4
  using OWAINS.SQL to install, 6 – 3
WHILE loops (PL/ SQL), 4 – 24
WLCTL. See Web Listener Control Utility

MAC003283

ORCL00941959

Reader's Comment Form

Name of Document: Oracle® WebServer User's Guide
Part No. A34986–2

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the topic, chapter, and page number below:

_____

_____

_____

_____

_____

_____

_____

_____

Please send your comments to:

Oracle Internet Products Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA 94065 U.S.A.
Fax: (415) 506–7200

If you would like a reply, please give your name, address, and telephone number below:

_____

_____

_____

Thank you for helping us improve our documentation.

MAC003284

Exhibit 45

US006651108B2

(12) **United States Patent**
Popp et al.

(10) **Patent No.: US 6,651,108 B2**
(45) **Date of Patent: *Nov. 18, 2003**

(54) **METHOD AND APPARATUS FOR GENERATING OBJECT-ORIENTED WORLD WIDE WEB PAGES**

(75) Inventors: **Nicolas Popp**, Menlo Park, CA (US); **Bruce Ong**, S.F., CA (US)

(73) Assignee: **Next Software, Inc.**, Redwood City, CA (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/515,057**

(22) Filed: **Aug. 14, 1995**

(65) **Prior Publication Data**

US 2002/0133637 A1 Sep. 19, 2002

(51) **Int. Cl.**[7] **G06F 9/44**

(52) **U.S. Cl.** ..... **709/315**; 709/330; 709/203; 709/219; 707/103 R; 715/513; 715/900; 715/908

(58) **Field of Search** 395/712, 680, 395/682, 772, 681, 687; 325/681; 709/303, 350, 316, 315, 330, 311, 217, 219, 900, 908, 911, 500, 501.1, 513, 517; 391/187

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,297,249	A		3/1994	Bernstein et al.
5,519,875	A	*	5/1996	Yokoyama et al. 709/316
5,600,831	A		2/1997	Levy et al.
5,623,656	A		4/1997	Lyons

(List continued on next page.)

OTHER PUBLICATIONS

Beck et al. "A Database Design for Uniform Representation of Hypermedia and Mathematical Models." ACM. 1994, p. 619–624.*

Bohm et al. "Storing HyTime Documents in an Object Oriented Database." ACM. 1994, p. 26–33.*

Bohm et al. "Structured Document Handling—a Case for Integrating Databases and Information Retrieval." ACM. 1994, p. 147–154.*

Meyer et al. "WAXweb: Toward Dynamic MOO–based VRML." ACM. May 1995, p. 105–108.*

Lenssen, Klaus. "Enabling Hypermedia Messaging BPRM—A Body Part Relationship Module." ACM. 1994, p. 157–163.*

An Object–Oriented Toolkt for ODA and HyperODA, H. Brown, F.C. Cole, E.A. Oxborrow, ACM, Oct. 1993.*

Turbo Vision 2.0 Programming Guide, Burland International, 1992 Prloiy Tu Interfax, Unix News, Sep. 1999, p. 20.*

TeleSoft Anonorus Teleuse Vision 2.1, Verwe, Larry, Bussiness Wire.*

Aronson, "HTML3 Manual of Style", pp. 44–45, 112–123, 1995.

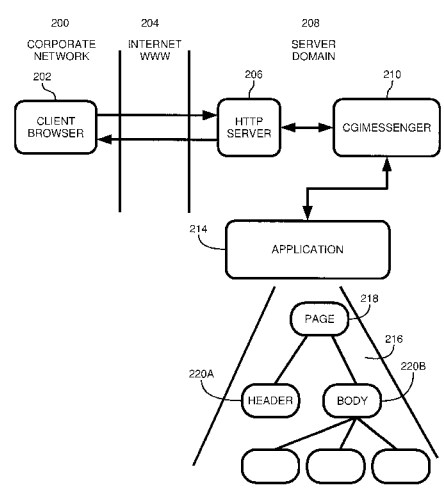
(List continued on next page.)

Primary Examiner—Sue Lao
Assistant Examiner—Lewis A. Bullock, Jr.
(74) *Attorney, Agent, or Firm*—The Hecker Law Group

(57) **ABSTRACT**

In the present invention, HTML elements are mapped to objects in an object-oriented environment. Classes of objects are defined for each HTML element as well as the HTML document (or page). By providing a one-to-one mapping between each HTML element and object classes, HTML documents can be manipulated programmatically. The properties of each element are stored in instance variables of the associated object. Each object class can include methods to manipulate the HTML element within an HTML document.

29 Claims, 9 Drawing Sheets



U.S. PATENT DOCUMENTS

5,652,879	A	*	7/1997	Harris et al. 707/101
5,655,130	A	*	8/1997	Dodge et al. 715/511
5,732,219	A	*	3/1998	Blumer et al. 709/227
5,742,762	A	*	4/1998	Scholl et al. 709/200
5,761,673	A	*	6/1998	Bookman et al. 709/311
5,905,248	A	*	5/1999	Russell et al. 235/462.15
5,930,512	A	*	7/1999	Boden et al. 717/102
5,956,736	A	*	9/1999	Hanson et al. 707/513
5,974,430	A	*	10/1999	Mutschler et al. 707/505
6,061,696	A	*	5/2000	Lee et al. 707/513
6,076,166	A	*	6/2000	Moshfeghi et al. 713/201
6,101,509	A	*	8/2000	Hanson et al. 707/513
6,112,242	A	*	8/2000	Jois et al. 709/225
6,163,878	A	*	12/2000	Kohl 717/100
6,182,092	B1	*	1/2001	Francis et al. 707/513
6,230,196	B1	*	5/2001	Guenthner et al. 709/223
6,249,291	B1	*	6/2001	Popp et al. 345/473
6,253,254	B1	*	6/2001	Erlenkoetter et al. 709/316
6,336,123	B2	*	1/2002	Inoue et al. 707/501.1

OTHER PUBLICATIONS

Electronic Mail Message: "Pricing Model—WebRex—TRUTH wanted!." Sep. 10, 1995, Ted Shelton, Information Technology Solutions, Inc.

Documents Posted on World Wide Web: "The WadsWorth Concept," last modified Aug. 10, 1995.

Documents Posted on World Wide Web: "Building Client/Server/Web Applications, How Fast Can Your Business Respond?", retrieved May 6, 1997.

Documents Posted on World Wide Web: "The Object–Oriented Development Enviornment For Creating Dynamic Web Applications," Visual Wave 2.0, retrieved May 5, 1997.

* cited by examiner

Figure 1

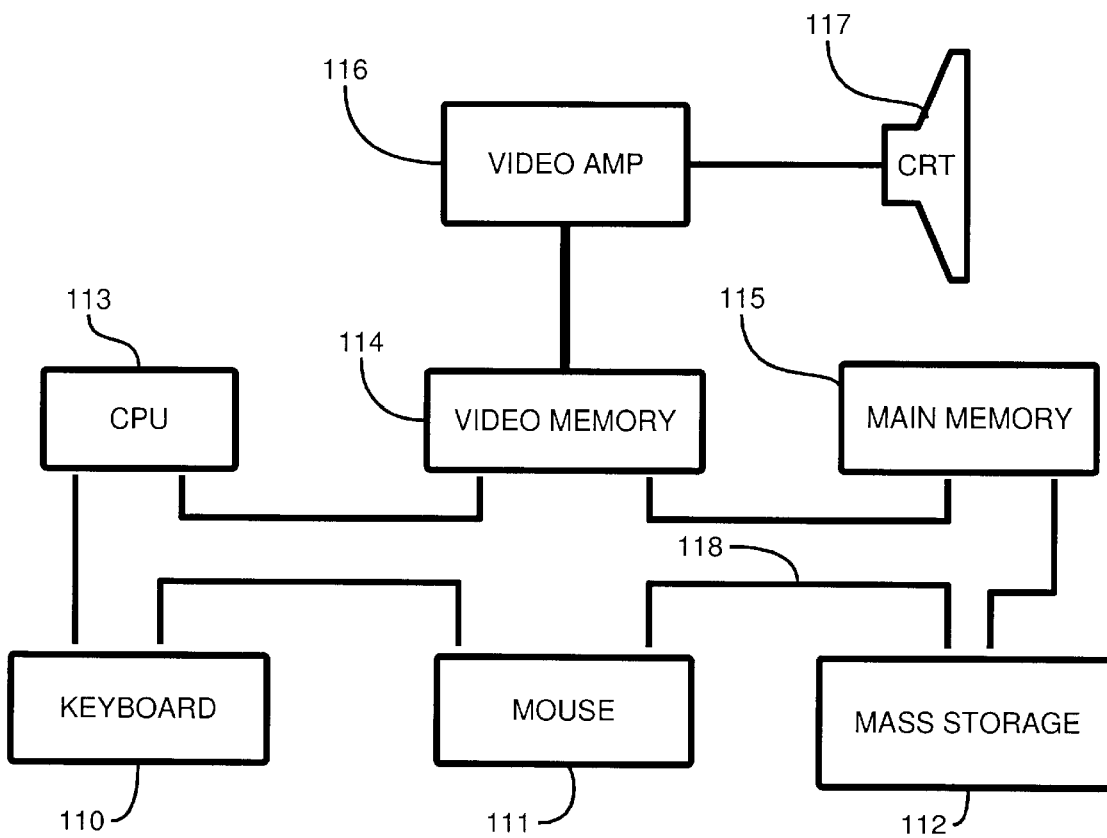

117
116
VIDEO AMP
CRT
113
114
115
CPU
VIDEO MEMORY
MAIN MEMORY
118
KEYBOARD
MOUSE
MASS STORAGE
110
111
112

Figure 2

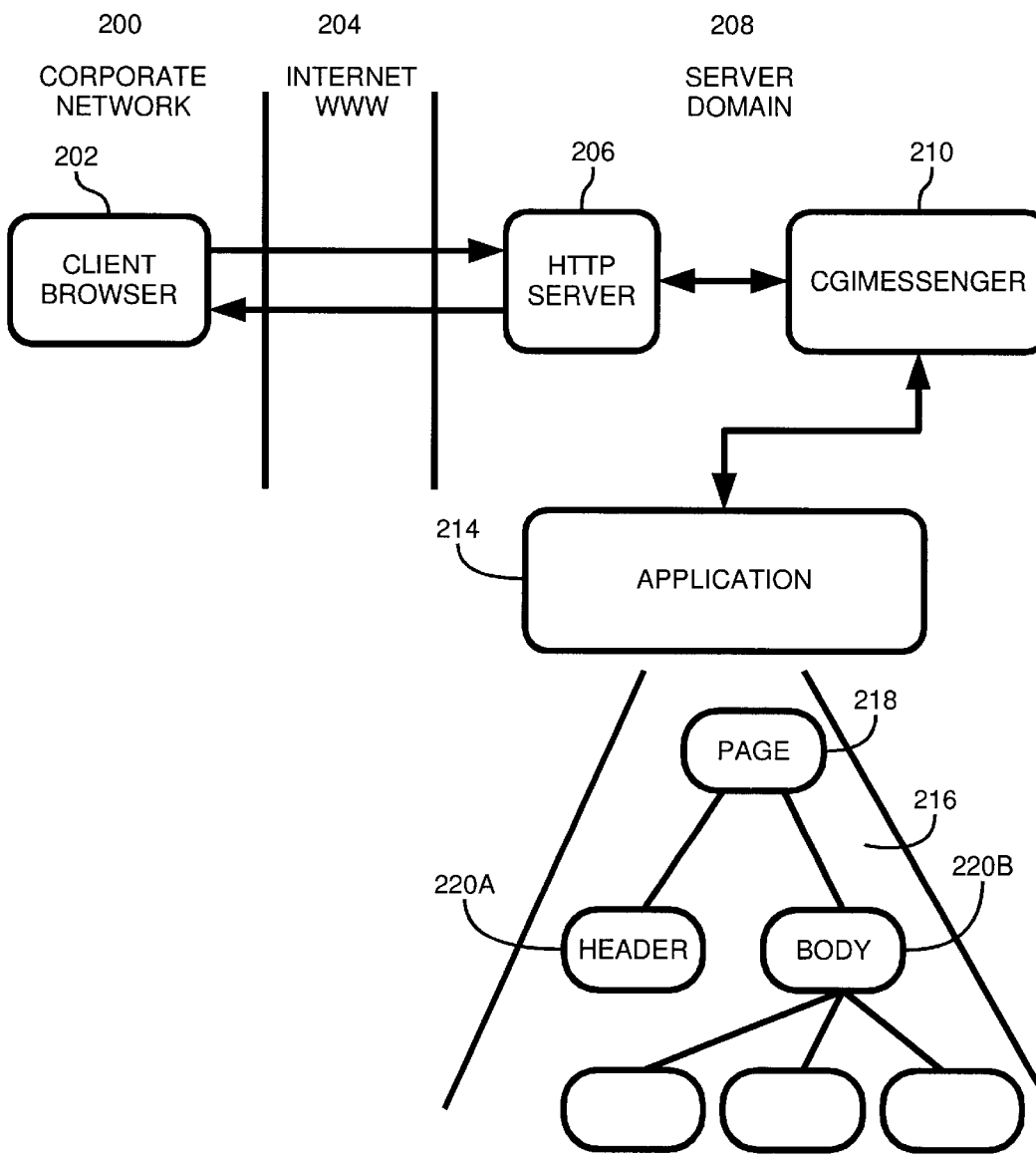

200
CORPORATE NETWORK
204
INTERNET WWW
208
SERVER DOMAIN
202
CLIENT BROWSER
206
HTTP SERVER
210
CGIMESSENGER
214
APPLICATION
218
PAGE
216
220A
220B
HEADER
BODY

Figure 3

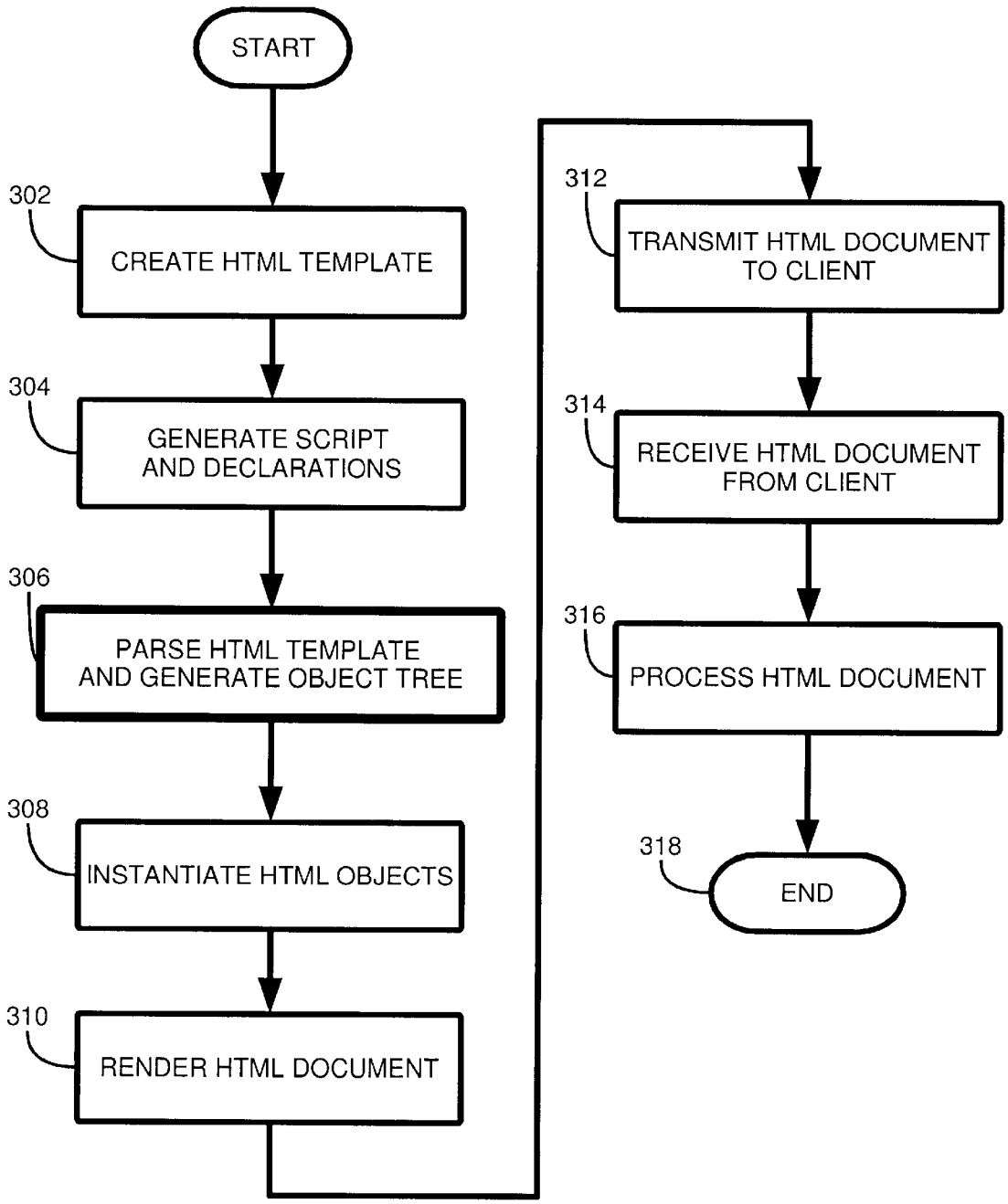

START
302
CREATE HTML TEMPLATE
304
GENERATE SCRIPT AND DECLARATIONS
306
PARSE HTML TEMPLATE AND GENERATE OBJECT TREE
308
INSTANTIATE HTML OBJECTS
310
RENDER HTML DOCUMENT
312
TRANSMIT HTML DOCUMENT TO CLIENT
314
RECEIVE HTML DOCUMENT FROM CLIENT
316
PROCESS HTML DOCUMENT
318
END

Figure 4

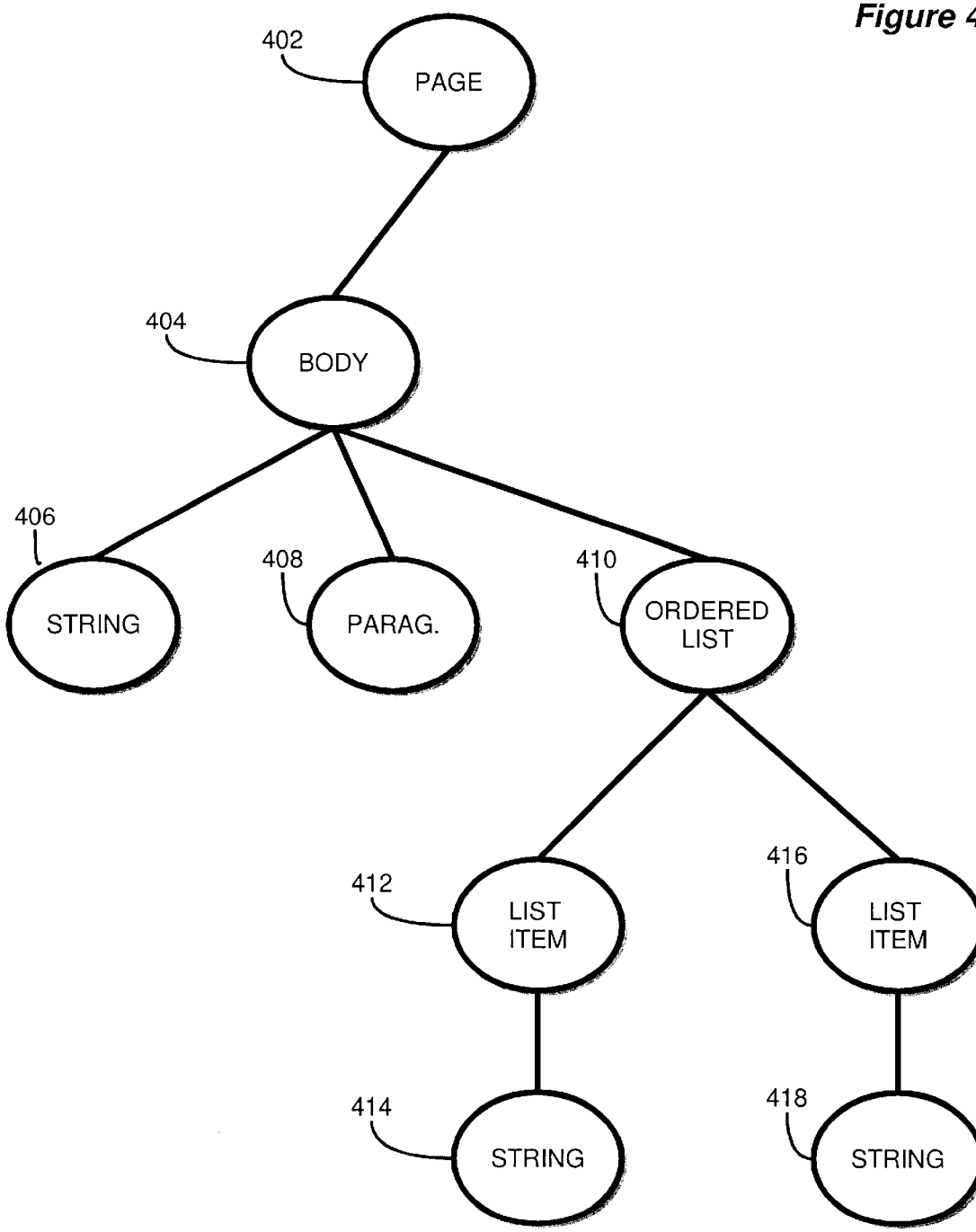

402
PAGE
404
BODY
406
STRING
408
PARAG.
410
ORDERED LIST
412
LIST ITEM
414
STRING
416
LIST ITEM
418
STRING

Figure 5

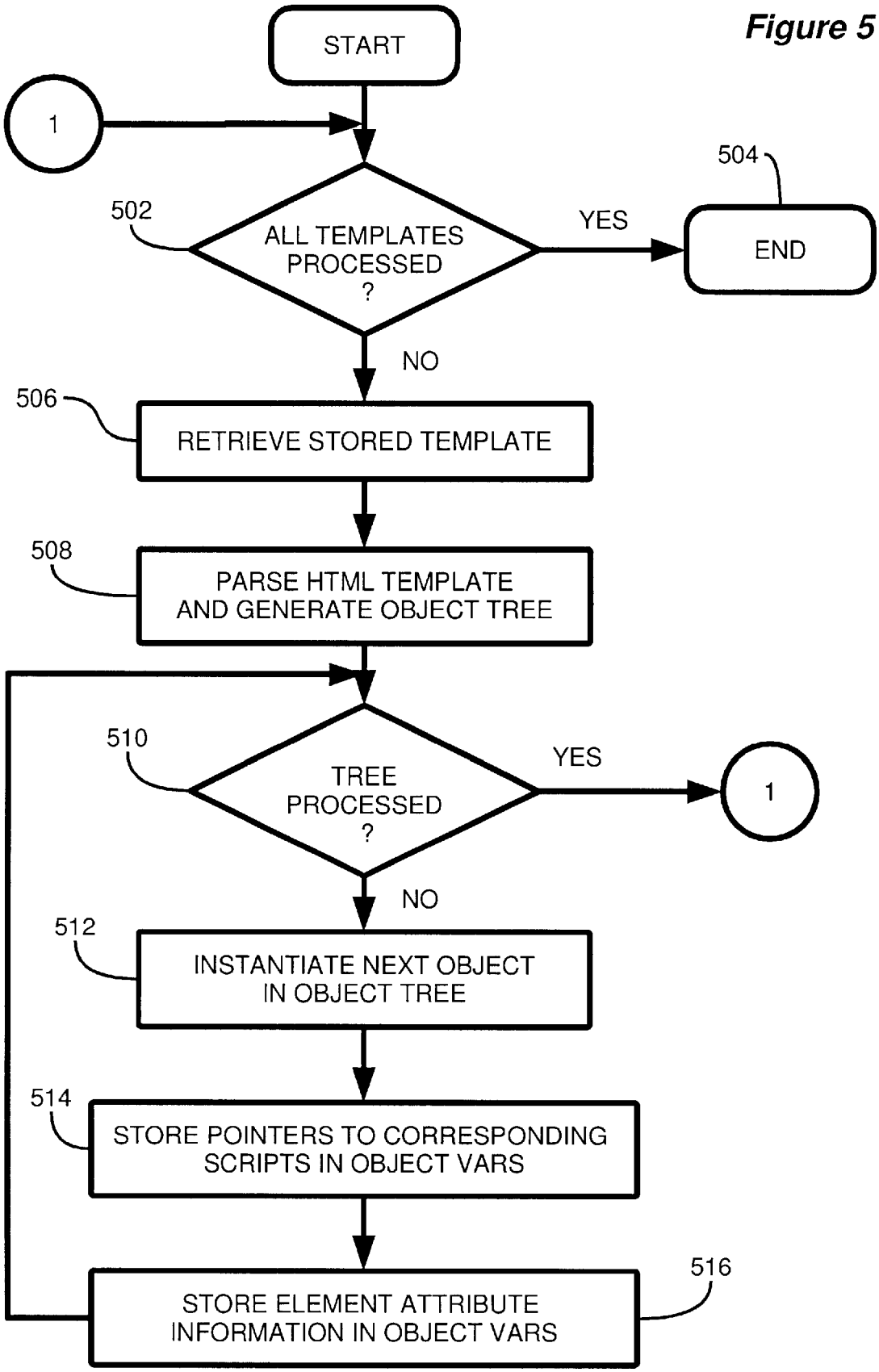

START
1
502
ALL TEMPLATES PROCESSED ?
YES
504
END
NO
506
RETRIEVE STORED TEMPLATE
508
PARSE HTML TEMPLATE AND GENERATE OBJECT TREE
510
TREE PROCESSED ?
YES
1
NO
512
INSTANTIATE NEXT OBJECT IN OBJECT TREE
514
STORE POINTERS TO CORRESPONDING SCRIPTS IN OBJECT VARS
516
STORE ELEMENT ATTRIBUTE INFORMATION IN OBJECT VARS

Figure 6A

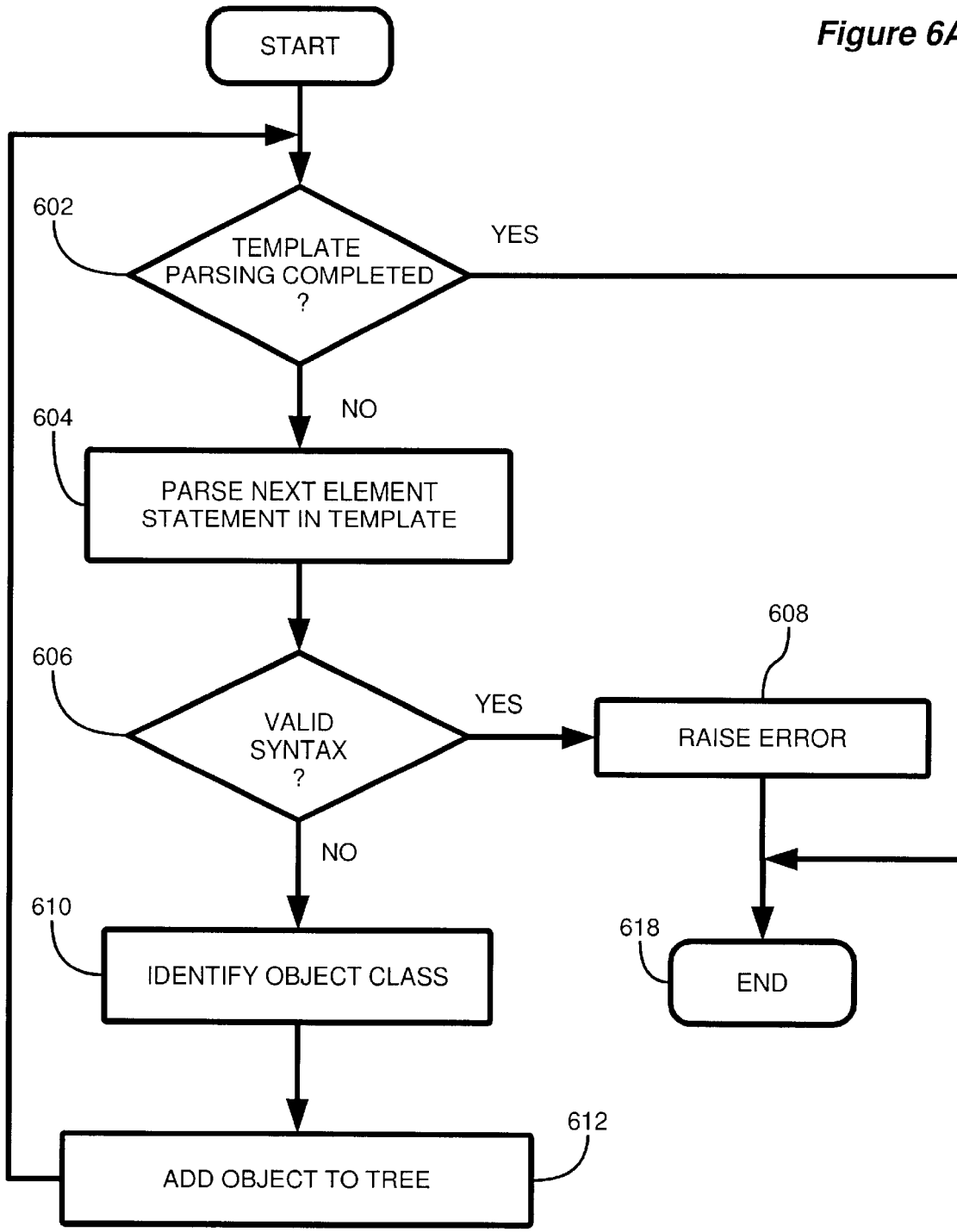

START
602
TEMPLATE PARSING COMPLETED ?
YES
NO
604
PARSE NEXT ELEMENT STATEMENT IN TEMPLATE
606
VALID SYNTAX ?
YES
608
RAISE ERROR
NO
610
IDENTIFY OBJECT CLASS
618
END
612
ADD OBJECT TO TREE

Figure 6B

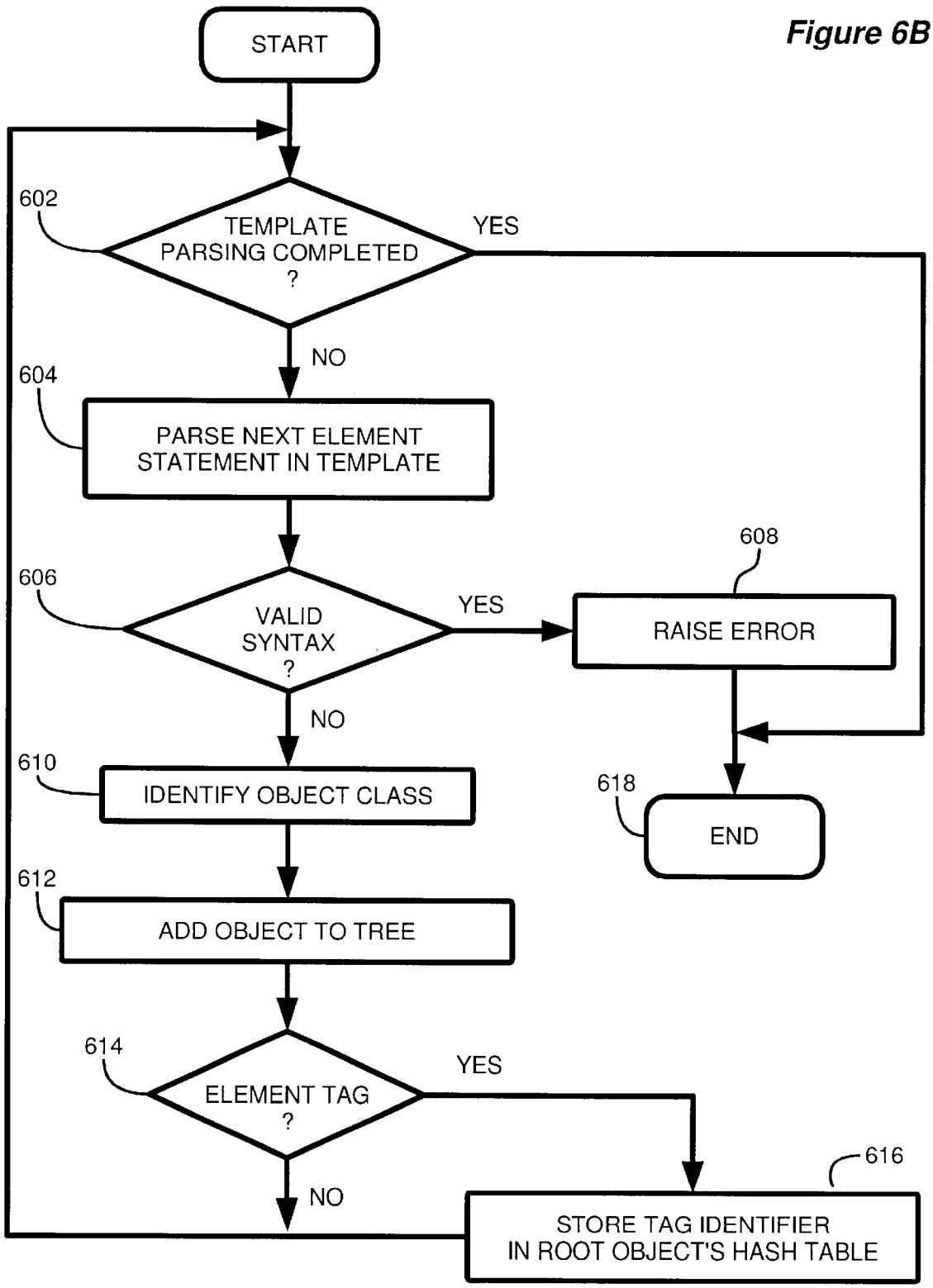

START
602
TEMPLATE PARSING COMPLETED ?
YES
NO
604
PARSE NEXT ELEMENT STATEMENT IN TEMPLATE
606
VALID SYNTAX ?
YES
608
RAISE ERROR
NO
610
IDENTIFY OBJECT CLASS
618
END
612
ADD OBJECT TO TREE
614
ELEMENT TAG ?
YES
NO
616
STORE TAG IDENTIFIER IN ROOT OBJECT'S HASH TABLE

Figure 7A

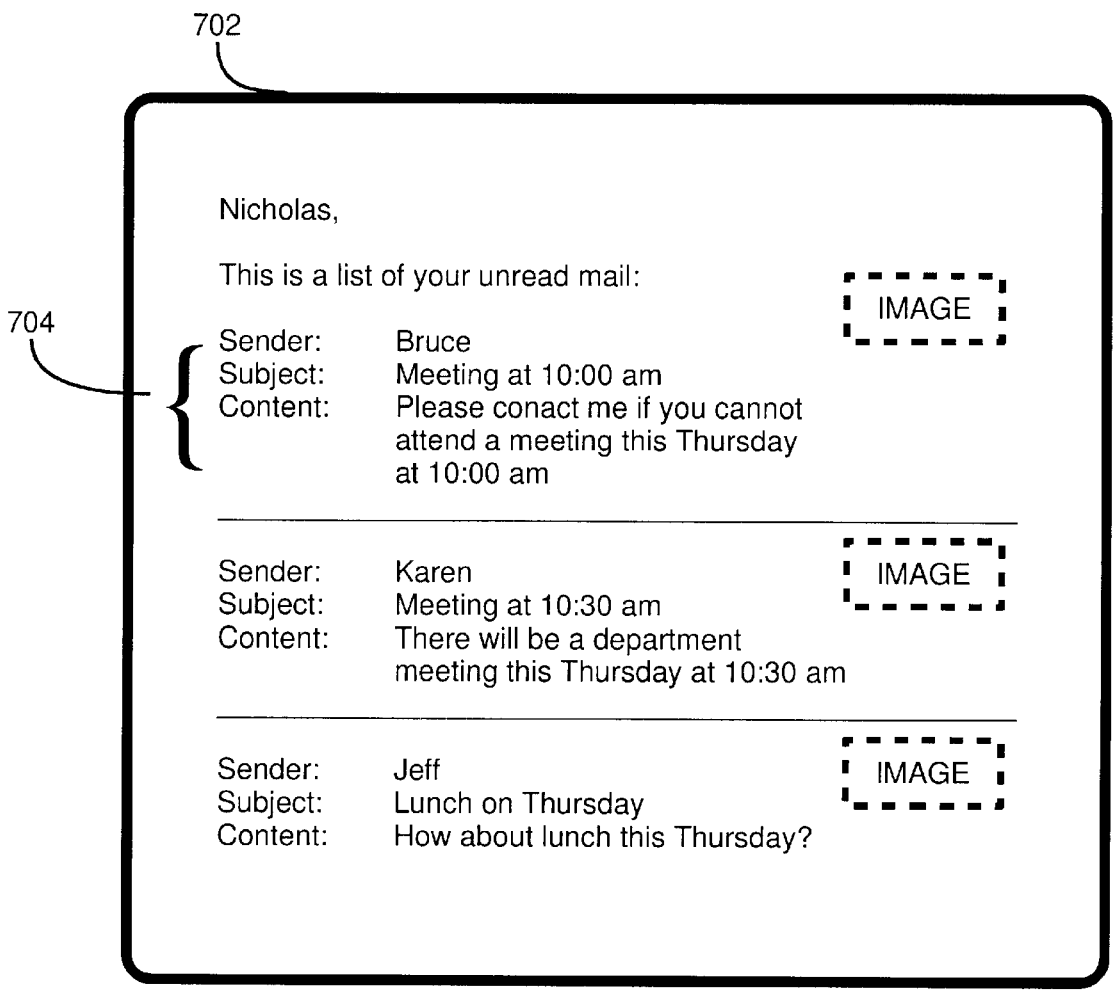
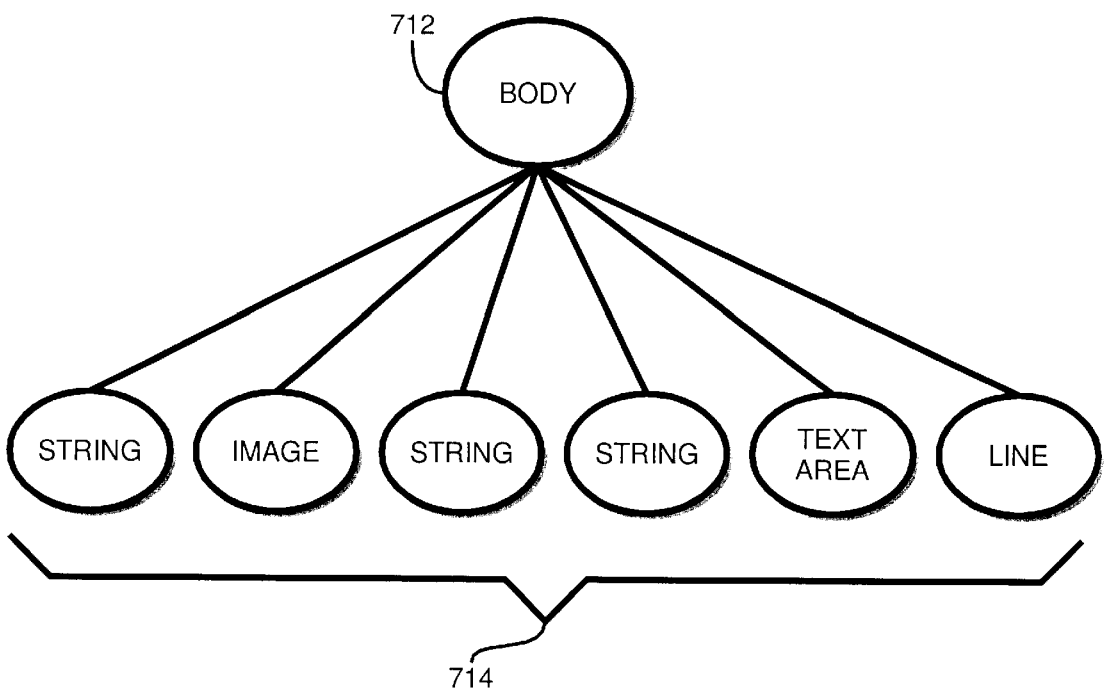

702
704
Nicholas,
This is a list of your unread mail:
IMAGE
Sender: Bruce
Subject: Meeting at 10:00 am
Content: Please conact me if you cannot attend a meeting this Thursday at 10:00 am
Sender: Karen
Subject: Meeting at 10:30 am
Content: There will be a department meeting this Thursday at 10:30 am
IMAGE
Sender: Jeff
Subject: Lunch on Thursday
Content: How about lunch this Thursday?
IMAGE

Figure 7B



METHOD AND APPARATUS FOR GENERATING OBJECT-ORIENTED WORLD WIDE WEB PAGES

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to the generation of object-oriented world wide web pages

2. Background

The Internet is a worldwide matrix of interconnected computers. An Internet client accesses a computer on the network via an Internet provider. An Internet provider is an organization that provides a client (eg., an individual or other organization) with access to the Internet (via analog telephone line or Integrated Services Digital Network line, for example). A client can, for example, download a file from or send an electronic mail message to another computer/client using the Internet.

To retrieve a file on the Internet, a client must search for the file, make a connection to the computer on which the file is stored, and download the file. Each of these steps may involve a separate application and access to multiple, dissimilar computer systems. The World Wide Web (WWW) was developed to provide a simpler, more uniform means for accessing information on the Internet.

The components of the WWW include browser software, network links, and servers. The browser software, or browser, is a user-friendly interface (i.e., front-end) that simplifies access to the Internet. A browser allows a client to communicate a request without having to learn a complicated command syntax, for example. A browser typically provides a graphical user interface (GUI) for displaying information and receiving input. Examples of browsers currently available include Mosaic, Netscape, and Cello.

Information servers maintain the information on the WWW and are capable of processing a client request. Hypertext Transport Protocol (HTTP) is the standard protocol for communication with an information server on the WWW. HTTP has communication methods that allow clients to request data form a server and send information to the server.

To submit a request, the client contacts the HTTP server and transmits the request to the HTTP server. The request contains the communication method requested for the transaction (e.g., GET an object from the server or POST data to an object on the server). The HTTP server responds to the client by sending a status of the request and the requested information. The connection is then terminated between the client and the HTTP server.

A client request therefore, consists of establishing a connection between the client and the HTTP server, performing the request, and terminating the connection. The HTTP server does not retain any information about the request after the connection has been terminated. HTTP is, therefore, a stateless protocol. That is, a client can make several requests of an HTTP server, but each individual request is treated independent of any other request. The server has no recollection of any previous request.

An addressing scheme is employed to identify Internet resources (e.g., HTTP server, file or program). This addressing scheme is called Uniform Resource Locator (URL). A URL contains the protocol to use when accessing the server (e.g., HTTP), the Internet domain name of the site on which the server is running, the port number of the server, and the location of the resource in the file structure of the server.

The WWW uses a concept known as hypertext. Hypertext provides the ability to create links within a document to move directly to other information. To activate the link, it is only necessary to click on the hypertext link (e.g., a word or phrase). The hypertext link can be to information stored on a different site than the one that supplied the current information. A URL is associated with the link to identify the location of the additional information. When the link is activated, the client's browser uses the link to access the data at the site specified in the URL.

If the client request is for a file, the HTTP server locates the file and sends it to the client. An HTTP server also has the ability to delegate work to gateway programs. The Common Gateway Interface (CGI) specification defines the mechanisms by which HTTP servers communicate with gateway programs. A gateway program is referenced using a URL. The HTTP server activates the program specified in the URL and uses CGI mechanisms to pass program data sent by the client to the gateway program. Data is passed from the server to the gateway program via command-line arguments, standard input, or environment variables. The gateway program processes the data and returns its response to the server using CGI (via standard input, for example). The server forwards the data to the client using the HTTP.

A browser displays information to a client/user as pages or documents. A language is used to define the format for a page to be displayed in the WWW. The language is called Hypertext Markup Language (HTML). A WWW page is transmitted to a client as an HTML document. The browser executing at the client parses the document and produces a displays a page based on the information in the HTML document.

HTML is a structural language that is comprised of HTML elements that are nested within each other. An HTML document is a text file in which certain strings of characters, called tags, mark regions of the document and assign special meaning to them. These regions are called HTML elements. Each element has a name, or tag. An element can have attributes that specify properties of the element. Blocks or components include unordered list, text boxes, check boxes, radio buttons, for example. Each block has properties such as name, type, and value. The following provides an example of the structure of an HTML document:

```
<HTML>
    <HEAD>
    . . . element(s) valid in the document head
    </HEAD>
    <BODY>
    . . . element(s) valid in the document body
    </BODY>
</HTML>
```

Each HTML element is delimited by the pair of characters "<" and ">" The name of the HTML element is contained within the delimiting characters. The combination of the name and delimiting characters is referred to as a marker, or tag. Each element is identified by its marker. In most cases, each element has a start and ending marker. The ending marker is identified by the inclusion of an another character, "/" that follows the "<" character.

HTML is a hierarchical language. With the exception of the HTML element, all other elements are contained within another element. The HTML element encompasses the entire document. It identifies the enclosed text as an HTML document. The HEAD element is contained within the

HTML element and includes information about the HTML document. The BODY element is contained within the HTML. The BODY element contains all of the text and other information to be displayed. Other HTML elements are described in an HTML reference manual.

The prior art HTML is not object-oriented. An HTML element is contained as a string within a flat, ASCII file. An application must be written to manipulate an HTML file. It would be beneficial to have the ability to map HTML elements to classes of objects that define the behavior of HTML elements.

SUMMARY OF THE INVENTION

In the present invention, HTML elements are mapped to objects in an object-oriented environment. Classes of objects are defined for each HTML element as well as the HTML document (or page). By providing a one-to-one mapping between each HTML element and object classes, HTML documents can be manipulated programmatically. The properties of each element are stored in instance variables of the associated object. Each object class can include methods to manipulate the HTML element within an HTML document.

An HTML document defines a World Wide Web page. An HTML document can be generated using HTML templates. Multiple HTML templates can be used to generate a single HTML document. An HTML template consists of HTML element statements. A parser parses an HTML template and generates an object tree.

The object tree is traversed during HTML document generation. When the HTML document is rendered, or generated, the root of the object tree is sent a message to create the document. The root object processes this message by creating its corresponding HTML element statement(s). The "create" message is then forwarded by the parent object to its children. This process is repeated until all of the objects in the object tree receive the message. Once all of the objects process the "create" message, the HTML document is generated.

An HTML document can also be generated without using an HTML template. An object tree can be created dynamically during runtime. For example, a root object (e.g., a PAGE object) is instantiated at runtime. During processing, additional HTML objects can be instantiated at inserted into the object tree. For example, a BODY object is inserted in the object tree as a child of the root object based on logic contained in an application procedure. Attributes of the BODY object can be set during application processing. Additional HTML objects can be added to the object tree in a similar manner until all of the objects have been assembled. As previously described, the HTML document is generated by sending a "create" message to the objects in the object tree.

An HTML template can include a server-side HTML extension known as the group extension. A group extension provides the ability to create a block of HTML statements. A name attribute of the group extension provides the ability to identify the group. The name attribute can be stored in a hash table. During processing, the group identity can be retrieved and used to traverse an object tree.

The group extension also ensures the scoping of named elements A group object maintains a hash table that includes the named elements which are its members. The group element scopes the named elements within itself. Therefore, two elements having the same name in different groups are distinguishable. Each one is scoped to its respective group.

A declarations file is used in combination with the group extension. The declarations file contains additional definition for a group extension. An entry in the declarations file includes a label that links the entry to the group extension. The entry also declares an HTML object. That is, the class of the HTML object is defined. Properties of the HTML object can also be defined within the entry. Values for properties provided in the object class definition can be used to populate the properties in an instance of the object class instantiated for the group.

A declaration entry modifies its associated group by adding element to the group or modifying the elements that already exist in the group. For example, an HTML object, or element, declared in the entry inserts itself into the group that bears the same name as the declaration entry. Property values that are declared in the declaration entry are used to modify the HTML object's properties.

An instance of the group extension contained in an HTML template is not included in the HTML document that is sent to the client browser. Therefore, the client browser does not need to recognize the group extension. When the HTML document is rendered, the HTML objects contained within a group object render themselves to generate the HTML for the group. The HTML objects generate actual HTML statements within the HTML document. That is, the group does not generate HTML statements. However, the group transmits a message to its children (e.g., an HTML object that inserted itself within the group) to generate HTML statements.

A declaration entry may contain, for example, a declaration for a text string HTML object. The STRING HTML object is inserted into the group identified by the entry. When the HTML document is rendered, the STRING HTML object generates the HTML statements necessary to insert the text string into the HTML document.

A group extension also provides the ability to identify a block of HTML as a repeating subcomponent of the HTML document. The block identified by the group extension can be repeated multiple times within the HTML document. A repeating group can be used to render HTML statements that contain data that is retrieved from an external data source, for example.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** provides an example of a general purpose computer to be used with the present invention.

FIG. **2** provides a block-level diagram that illustrates components of the present invention.

FIG. **3** provides an example of a flow for processing a Web page using the object-oriented capabilities of the invention.

FIG. **4** provides a diagram of the object tree or hierarchy for an HTML template provided in Table One.

FIG. **5** provides an example of a flow for processing stored templates for an HTML document.

FIG. **6**A provides an example of a process flow to parse a template and generate an object tree.

FIG. **6**B modifies the flow provided in FIG. **6**A to include steps to add tag name to a hash table.

FIG. **7**A provides an example of a Web page that contains a repeating element.

FIG. **7**B provides an example of the object tree for a repeating subcomponent of a Web page.

DETAILED DESCRIPTION OF THE INVENTION

A method and apparatus for generating object-oriented World Wide Web pages is described. In the following

description, numerous specific details are set forth in order to provide a more thorough description of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well-known features have not been described in detail so as not to obscure the invention.

The present invention can be implemented on a general purpose computer such as illustrated in FIG. **1**. A keyboard **110** and mouse **111** are coupled to a bi-directional system bus **118**. The keyboard and mouse are for introducing user input to the computer system and communicating that user input to CPU **113**. The computer system of FIG. **1** also includes a video memory **114**, main memory **115** and mass storage **112**, all coupled to bi-directional system bus **118** along with keyboard **110**, mouse **111** and CPU **113**. The mass storage **112** may include both fixed and removable media, such as magnetic, optical or magnetic optical storage systems or any other available mass storage technology. Bus **118** may contain, for example, 32 address lines for addressing video memory **114** or main memory **115**. The system bus **118** also includes, for example, a 32-bit DATA bus for transferring DATA between and among the components, such as CPU **113**, main memory **115**, video memory **114** and mass storage **112**. Alternatively, multiplex DATA/address lines may be used instead of separate DATA and address lines.

In the preferred embodiment of this invention, the CPU **113** is a 32-bit microprocessor manufactured by Motorola, such as the 680×0 processor or a microprocessor manufactured by Intel, such as the 80×86, or Pentium processor. However, any other suitable microprocessor or microcomputer may be utilized. Main memory **115** is comprised of dynamic random access memory (DRAM). Video memory **114** is a dual-ported video random access memory. One port of the video memory **114** is coupled to video amplifier **116**. The video amplifier **116** is used to drive the cathode ray tube (CRT) raster monitor **117**. Video amplifier **116** is well known in the art and may be implemented by any suitable means. This circuitry converts pixel DATA stored in video memory **114** to a raster signal suitable for use by monitor **117**. Monitor **117** is a type of monitor suitable for displaying graphic images.

The computer system described above is for purposes of example only. The present invention may be implemented in any type of computer system or programming or processing environment. When a general purpose computer system such as the one described executes the processes and process flows described herein, it is a means for generating an HTML page using object-oriented HTML elements.

In the present invention, HTML elements are mapped to objects in an object-oriented environment. Classes of objects are defined for each HTML element as well as the HTML document (or page). Each HTML element can be mapped to a corresponding object class (e.g., Objective C, C++, or SmallTalk object class). By providing a one-to-one mapping between each HTML element and object classes, HTML documents can be manipulated programmatically. The properties of each element are stored in instance variables of the associated object. Each object class can include methods to manipulate the HTML element within an HTML document.

The object-oriented approach used by the present invention provides the ability to dynamically generate a page, or HTML document. Some or all of a page can be generated based on input received in a receive form, generated at runtime, or retrieved from an external data source (e.g., database or electronic mail system). The dynamic generation of HTML documents is more fully described in a co-pending U.S. patent application entitled "Method and Apparatus for Processing Internet Transactions", Ser. No. 08/532,491, U.S. Pat. No. 6,249,291, filed Sep. 22, 1995 and incorporated herein by reference.

As previously indicated, the present invention objectifies (Objective C, C++ objects, SmallTalk objects, etc.) HTML elements. By doing so, the HTML objects can be connected to other objects. That is, the HTML objects can be connected to objects in the business model (i.e., data model objects). Internet transaction processing is more fully described in a co-pending U.S. patent application entitled "Method and Apparatus for Processing Internet Transactions ", Ser. No. 08/532,491, filed U.S. Pat. No. 6,249,291 Sep. 22, 1995 and incorporated herein by reference.

The present invention provides the ability to access an application using any interface. For example, a client can access the same application via the Internet using a system running Windows, MAC/OS, Sun OS, NextStep, etc. Referring to FIG. **2**, client **202** is resident on a corporate network **200**. Corporate network **200** is a local area network comprised of personal computers such as client **202**, for example. Client **202** executes a browser such as Netscape or Mosaic to interface with World Wide Web (WWW) **204**. The browser executing on client **202** displays HTML pages transmitted to client **202** via WWW **204**. When the client browser submits a request contained within an HTML page, the page is transmitted via WWW **204** to HTTP server **206**. One attribute of the submitted page can be a Universal Resource Locator (URL). A URL specifies procedures to be executed and/or it data input by the user, for example.

HTTP server **206** can service the client request by, for example, retrieving a return form or file (specified by a URL contained in the HTML document) and transmitting it to client **202** via WWW **204**. Further, HTTP server **206** can initiate a program identified in the URL. In the preferred embodiment, the URL specifies CGIMessenger **210** to service the client request. CGIMessenger is a Common Gateway Interface (CGI) script. The following provides an example of a URL:

http://www.next.com/cgi_bin/cgimessenger/app_name/rcv_form/send_form

A URL has multiple parts. The first part, "http:" specifies the protocol to be used. Other examples of protocol include "ftp:", "gopher:" and "wais". The next part specifies the Internet domain name of the server. In this case, the Internet domain name of the HTTP server is "www.next.com". The next part, "/cgi_bin/cgi_messenger" specifies the path and filename of the desired resource. The resource can be a file, form, or program, for example. In this case, the resource is a common gateway interface program that is used to route requests to applications. The application that processes the client request is specified in the URL with the "app_name" specification. The names of the receiving form (e.g., the form containing the client request) and the sending form (e.g., a reply to the client request) can be included in the URL.

A client request can result in the invocation of application **214** in the server domain **208** for example. The client request is received by the HTTP server **206**. The HTTP server **206** initiates CGIMessenger **210**. To service the client request, CGIMessenger **210** communicates with HTTP server **206** to obtain information regarding the client request. For example, CGIMessenger **210** obtains the information that accompanied the user request such as form and/or applica-

tion names and/or user input. The form names can include the name of the form that was submitted to initiate the client request as well as a return form.

The application name identifies the application that services the client request. If an application is specified, CGIMessenger **210** transmits the client request and corresponding information transmitted from HTTP Server **204** to application **214**. Application **214** executes an interaction flow to satisfy the user request.

A simple hypertext request (e.g., a request for another form specified in the request) is serviced by HTTP Server **206** by returning the specified form. For requests other than simple hypertext requests, the present invention can execute logic such as CGIMessenger **210** and application **214**, for example, to process a client request. The application includes objects such as those created in an objective C, C++, or SmallTalk environment to manage the components of an HTML document (i.e., HTML elements). The behavior defined by the objects manipulate the HTML elements based on the information available to the objects, for example.

Preferably, a user interface (UI) is used to define the HTML elements of a Web page. However, HTML elements can be defined using a text editor. The output of either a UI tool or a text editor is an ASCII, or flat, file that contains definitions for one or more HTML elements. The HTML elements are arranged within the file in the definitional hierarchy (e.g., BODY element is contained within the HTML element).

The HTML elements that define a Web page can be included in a single ASCII file. A file that contains HTML elements is referred to as an HTML template. An HTML template can include a complete definition of the Web page. However, an HTML template may define a subset of the Web page. Two or more HTML templates that define portions of the Web page can be combined to provide a complete definition. The following provides an example of the contents of an HTML template:

TABLE ONE

```
<HTML>
    <BODY>
    This is a list of employees:
    <P>
    <OL>
    <LI> James </LI>
    <LI> Mary </LI>
    </OL>
    </BODY>
</HTML>
```

The above HTML template includes HTML, BODY, Paragraph (P), Ordered List (OL),and List Item (LI) elements. The present invention defines a class for each HTML element. An object class defines instance variables to store information associated with the HTML element such as attributes, or properties, of an HTML element. Methods can be defined to manipulate an HTML element to render (i.e., generate) an HTML document, for example.

Referring to FIG. **2**, application **214** includes objects **216**. Objects **216** correspond to the HTML elements that define a WWW page. Objects **216** are arranged in a tree structure that corresponds to the hierarchical structure of the HTML elements that they implement. To render an HTML document, application **214** sends a message to the root object **218** to render itself. In response to this message, root object **218** renders the corresponding HTML element(s). Root object **218** sends a message to its children, objects **220**A–**220**B, to render themselves. This process is continued until all of the nodes in the object tree have been processed.

The objects in an object tree are instantiated based on pre-defined object classes. Object classes are defined to manipulate and generate HTML elements in an HTML document. The name of the object class can mirror the names of the corresponding HTML element to provide an easy association between the two. Alternatively, any mapping scheme can be used to map an HTML element to an object class. Instance variables can be used to store property information defined by properties in the HTML element statement. An object class can further contain instance variable declarations to store other information associated with an HTML element.

An object class can have associated method declarations that can be used to manipulate the HTML document. For example, an object class may have a method that varies the display characteristics of the HTML element based on the outcome of some Boolean logic. Further, instance variables can be used to provide a pointer to scripts used with the HTML document.

A script is a type of program that consists of a set of instructions that is read and interpreted by another application. An instruction is an expression that conforms to the rules and syntax expected by a script interpreter. A script language (set of script instructions) may include simple control statements such as loops and if/then expressions.

Since the language does not include complex constructs, the language is easy to master. As with other interpretive languages, a developer does not have to perform steps to translate a script into machine-readable code. Even though a script is interpreted at runtime, it is not considered to cause any degradation in the overall response time (the actual network response times are the primary concern). Therefore, scripts are typically used to provide the logic associated with a Web page.

Appendix A provides a list of some object classes for HTML elements. Additional or modified class definitions can also be used with the present invention. A library of object classes are preferably predefined.

FIG. **3** provides an example of a flow for processing a Web page using the object-oriented capabilities of the invention. At step **302**, a Web page developer creates the HTML templates to provide a Web page definition. Alternatively, the Web page can be defined dynamically at runtime without using an HTML template. In this case, there is no need for the developer to perform step **302**. The developer generates a script and declarations files needed to process the Web page at step **304**. A declarations file contains information that is used to generate the Web page. The HTML template (s), script(s), and declarations file(s) are stored for use when the Web page definition is to be sent to a client browser.

When the Web page definition is requested by a client browser, the associated HTML template(s), script(s), and declarations are retrieved from storage. At step **306**, a parser is invoked to parse the statements contained in the HTML template(s). HTML elements are identified by the parser. An object tree is generated based on the HTML elements identified and the hierarchical structure inherent in HTML and defined in the HTML template(s).

An HTML object is instantiated for each of the HTML elements defined in the HTML template(s) at step **308**. At step **310**, the objects instantiated in step **308** are used to render the Web page definition (i.e., HTML document). The HTML document is sent to the client browser at step **312**. The client browser interprets the HTML document and

displays the Web page. When the client submits the page, the page is returned to the HTTP server as an HTML document. The HTML document is received at step **314**. At step **316**, the HTML objects are used to perform the processing based on the client input. Processing ends at step **318**.

The object tree generated during parsing identifies the objects and hierarchy defined in an HTML template. FIG. **4** provides a diagram of the object tree or hierarchy for the HTML template provided in Table One.

Page object **402** corresponds to the HTML element. The BODY element is contained within the HTML element hierarchy. This hierarchy is represented in the object tree by a parent-child relationship. Body object **404** is a child of its parent, page object **402**. Similarly, string object **406**, paragraph object **408**, and ordered list object **410** are children of their parent, body object **404**. String object **406** corresponds with the "This is the list of employees:" text string in Table One. Paragraph object **408** and ordered list object **410** correspond with the paragraph and ordered list elements, respectively, in the HTML document. Below the ordered list element is a pairing of a list item element and string element in the HTML document in Table One.

Before an HTML document is sent to the client, the HTML objects render the document. An HTML document consists of HTML statements. HTML statements define HTML elements. An HTML document defines the components of a Web page. The HTML document is sent to client browser **202** from HTTP Server **206**, for example. Browser **202** identifies the components of the Web page by parsing the HTML document and generates the Web page.

Before the HTML document is transmitted to client browser **202** by HTTP server **206**, it is generated using the object-oriented capabilities of the present invention. As previously indicated, an HTML document is defined using HTML templates created using a process such as the one illustrated by FIG. **3**. FIG. **5** provides an example of a flow for processing stored templates for an HTML document.

At step **502**, a determination is made whether all templates needed to define the HTML document have been processed. If all of the templates have been processed, processing ends at step **504**. If all of the HTML templates have not been processed, processing continues at step **506** to retrieve the next stored template. At step **508**, the template is parsed and an object tree is generated. Each object identified in the object tree corresponds to an HTML element defined in the HTML template. Each object in the tree corresponds to an object class. For each element in the HTML template, an instance of the corresponding object class is instantiated. At step **510**, a determination is made whether all of the objects in the template's object tree have been processed. If they have, processing continues at step **502** to determine whether all of the HTML document's templates have been processed.

If the template's object tree has not been processed, processing continues at step **512** to instantiate the next object in the object tree. At step **514**, pointers to a script can be set in the instantiated object's instance variables, for example. Element attribute information is stored in the object's instance variables at step **516**. Processing continues at step **510** to process any remaining portion of the object tree.

As illustrated in FIGS. **3** and **5**, a parsing routine is used at HTML document generation time. Any parser can be used with the present invention. Parsers and parsing routines can be downloaded from various sites on the Internet. The parsing process is performed during document generation to retrieve HTML element information and to generate an object tree. FIG. **6**A provides an example of a process flow to parse a template and generate an object tree.

At step **602**, a determination is made whether the template has been parsed. If it has, processing ends at step **618**. If the template has not been completely parsed, processing continues at step **604** to parse the next element definition in the template. If, at step **606**, a determination is made whether a syntax error exists in the element definition, processing continues at step **608** to raise an error condition. If an error occurs, processing ends at step **618**. Alternatively, processing can continue to parse the rest of the template.

If no syntax error occurred, the parser proceeds to identify the type of element defined by the current element statement. At step **610**, the object class that corresponds to the element type is identified. The object is added to the object tree at step **612**. Processing continues at step **602** to determine if additional parsing is necessary.

An HTML document can also be generated without using an HTML template. An object tree can be created dynamically during runtime. PAGE object **402** is instantiated at runtime, for example. During processing, additional HTML objects can be instantiated and inserted into the object tree as children, grandchildren, great-grandchildren, etc. of PAGE object **402**. For example, BODY object **404** is instantiated at runtime based on logic contained in an application procedure. BODY object **404** is inserted in the object tree as a child of PAGE object **402**. Attributes of the BODY object can be set in the application processing. The remaining portion of the object tree in FIG. **4** can be created dynamically as well.

An object tree that is created dynamically without using HTML templates, or one that is created based on one or more HTML templates can be modified dynamically at runtime. Once the object tree is generated, the HTML statements of an HTML document are generated by sending a "create" message to the objects in the object tree.

There is a standard set of HTML elements that are understood by most of the Web browsers. Some Web browsers offer additional HTML elements or extensions. These extensions provide additional functionality. However, because extensions are not part of the standard HTML, one browser's HTML extension may not be interpretable by another browser.

The extensions offered by client browsers are interpreted on the client side. The present invention provides an extension to HTML that is used on the HTTP server side. The HTML extension is filtered out before a Web page is sent to a client browser. It is used to interpret an HTML template and to render an HTML document before is transmitted to the client browser. Because HTML extensions are not sent to the client browser, there is no need for a client browser to support these extensions. Thus, existing client browsers that support the standard HTML can be used to display HTML documents using the present invention.

The present invention uses an extension to the standard HTML known as the group extension The group extension provides the ability to combine HTML elements or statements in a single block. A name attribute of the group extension provides the ability to identify the block, or group, by name. The name attribute can be stored in a hash table. During processing, the group identity can be retrieved and used to traverse an object tree.

The group extension also ensures the scoping of named elements (i.e., elements that have a name by virtue of a name attribute). A group object maintains a hash table that

includes the named elements which are its members. The group element scopes the named elements within itself. Therefore, two elements having the same name in different groups are distinguishable. Each one is scoped to its respective group. For example, a group element named "Nico" contained within a parent group named "Employee" is distinct from a group named "Nico" within a parent group named "Tourist".

A declarations file is used in combination with the group extension. The declarations file contains additional definition for a group extension. An entry in the declarations file includes a label that links the entry to the group extension. The entry also declares an HTML object. That is, the class of the HTML object is defined. Properties of the HTML object can also be defined within the entry. Values for properties provided in the object class definition can be used to populate the properties in an instance of the object class instantiated for the group.

A declaration entry modifies its associated group by adding elements to the group or modifying the elements that already exist in the group. For example, an HTML object, or element, declared in the entry modifies the group by inserting itself into the group that bears the same name as the declaration entry. Property values that are declared in the declaration entry are used to modify the HTML object's properties.

An NSWTAG marker is used to identify the bounds of the group. The following is an example of a format for an HTML group extension statement using the NSWTAG marker:

<NSWTAG NAME="GREETING"></NSWTAG>

In this example, the group, or NSWTAG extension, does not contain an HTML statement. The NSWTAG extension is identified by the <NSWTAG> and </NSWTAG> delimiters. The NAME portion of the extension specifies the name of the group. This name can be used to link to a declaration in a declaration file. An example of an entry contained in an external definition for "GREETING" is as follows:

GREETING:NSWString {value="Welcome to the Cyber Surfshop"}

The group extension is associated with the declaration file entry using the name attribute in the NSWTAG extension statement. The same name is provided as a label in the declaration file. The label field of an entry is the first field in the entry and is located to the left of the colon (e.g., "GREETING). To the right of the label (after the colon) in the declaration file entry is an indication of an object class. The object class field specifies the class of HTML object that is to be instantiated and inserted in the "GREETING" group. The "NSWString" specification in the "GREETING" entry results in the creation of an instance of the "NSWString" HTML object class. The NSWString class provides object cover for free-floating text. The "NSWString" HTML object instance inserts itself into the "GREETING" group.

The declaration contains a property field (i.e., delimited by brackets). The property field, value, is used to initialize a property in the "NSWString" HTML object. The value field contains the text that is to be inserted into the HTML document. When the "GREETING" group portion of the object is rendered, a "create" message is sent to the "GREETING" group object. It does not generate any HTML statements in the HTML document. However, it sends a "create" message to its children. As its child, the "NSWString" HTML object receives the "create" message. The "NSWString" HTML object inserts the text contained in the value property into the HTML document.

The group extension can be used to insert a block of HTML statements into an HTML document. The block of HTML statements can occur once in the HTML template. The block can also be repeated within the HTML document. FIG. **7**A provides an example of a Web page that contains block that is repeated (i.e., a repeating subcomponent).

Web page **702** provides a list of a client's unread electronic mail. As displayed in Web page **702**, a mail item **704** contains information about the sender (name and image), subject, and content of the mail message. Mail item **704** is repeated for each piece of electronic mail. A horizontal rule separates each mail item. The sender, subject, content and horizontal rule elements constitute the repeating subcomponent of Web page **702**.

A group element acts outer block of a group of HTML elements that generate the HTML statement block (either repeating or nonrepeating). In an object tree, the group object acts as the root for the HTML objects that correspond to the HTML elements. FIG. **7**B provides an example of the object tree that is symbolic of the repeating subcomponent of Web page **702** of FIG. **7**A. The group object **712** is the parent of the objects **714** contained within the group. Each of the objects **714** contained within the group are children of group object **712**. The children of group object **712** (i.e., objects **714**) can be repeated to handle the number of mail items to be displayed to the client.

In addition to defining the root of a repeating component, the group extension can be used to define the root of a subtree. The subtree can be combined with other trees or subtrees to generate a complete object. The combined object tree can be used to render a Web page. Thus, for example, Web page **702** may be defined in two separate HTML templates. One template defines the repeating portion and the other defines the non-repeating portion. FIG. **7**B provides the object tree for the HTML template that defines the repeating portion of Web page **702**. The group extension can be defined in both templates such that the two templates can be combined to generate a single HTML document.

Group extension scan be nested in the same way that HTML elements are nested within each other. An outer group provides scoping for an inner group. Therefore, a parent group object provides scoping for its children. Table Two provides an example of HTML that includes an NSWTAG extension.

TABLE TWO

```
<HTML>
    <NSWTAG NAME="SELECT_FORM">
        <FORM>
                    . . .
            <NSWTAG NAME="SAILBOARDS">
                <NSWTAG NAME="CHECK"><INPUT TYPE=
                  "checkbox"></NSWTAG>
                <NSWTAG NAME="NAME"></NSWTAG>
            </NSWTAG>
                    . . .
        </FORM>
    </NSWTAG
                    . . .
</HTML>
```

The first NSWTAG extension provides an identifier for the block containing the form element. The NSWTAG statement includes a name attribute, "SELECT_FORM", to identify this block, or group. Within the "SELECT_FORM" block, a "SAILBOARDS" NSWTAG extension provides an identifier for a repeating subcomponent of the HTML. There are two NSWTAG extensions within the repeating subcom-

ponent (i.e., "CHECK" and "NAME"). An input HTML element is nested within each of these NSWTAG extensions.

The declarations file that is associated with the HTML in Table Two contains further definition for the NSWTAG extensions. For example, the outer NSWTAG extension (i.e., "SELECT_FORM") has the following associated entry in the declarations file:

SELECT_FORM:NSWForm {action=onSelection()};

The label portion of the declaration file entry (i.e., "SELECT_FORM") links the entry with the NSWTAG extension in the HTML template having the same name. The next field in the entry (i.e., located to the right of the colon) indicates that an instance of the NSWForm object class should be instantiated and inserted within the SELECT_FORM group object. Following the element/object specification, the entry includes a value for a property or attribute of the NSWFORM object. That is, the action attribute is assigned the value of "onSelection()". The "onSelection()" value indicates the procedure to invoke in the script associated with the HTML document when an action is taken by the client, for example.

The NSWTAG extension named "SAILBOARDS" is contained within the SELECT_FORM group. It also begins a repeating subcomponent. An example of the declarations file entry associated with the "SAILBOARDS" NSWTAG extension is as follows:

SELECT_FORM SAILBOARDS NSWRepetition {list=sailboards; item=sailboard,

The label in the declarations file entry specifies that the entry is associated with the "SAILBOARDS" group that is within the SELECT_FORM group. The entry further indicates that the "SAILBOARDS" group is a repetition group. The "SAILBOARDS" group can cause a block of HTML statements to be repeated in the HTML document. The NSWRepetition specification in the entry indicates that an repetition object should be instantiated to process the repeating subcomponent. The repetition object can contain HTML objects that generate a multiple occurrences of an HTML statement block. The number of times, N, that the block is repeated is based on data that is to be loaded into the HTML (e.g., list of sailboards).

The list attribute specified in the "SELECT_FORM.SAILBOARDS" declarations file entry points to a container object (i.e., object that holds other objects), "sailboards". The item attribute points to the information to be retrieved to populate the repeating subcomponent for the "SAILBOARDS" repetition group.

The HTML template and the declarations file contains entries that further define the "SELECT_FORM.SAILBOARDS" repetition group. Two group extensions are contained within the "SELECT_FORM.SAILBOARDS" nested group structure in the HTML template. These groups, "CHECK" and "NAME", have the following declarations, respectively:

SELECT_FORM.SAILBOARDS.CHECK:NSWInput {value=sailboard, return=selectionArray};

SELECT_FORM.SAILBOARDS.NAME NSWString {value=sailboard},

An object of class "NSWInput" is instantiated based on the specification in the "CHECK" NSWTAG statement and the "SELECT_FORM.SAILBOARDS.CHECK" declaration file entry. It inserts itself within the "SELECT_FORM.SAILBOARDS.CHECK" nested group.

Preferably, property values of the newly instantiated HTML object are retrieved from its corresponding declaration. However, property information can be defined in the HTML template. For example, the "SELECT_FORM.SAILBOARDS.CHECK" group HTML element in the HTML template specifies the type property (i.e., "text") for the NSWInput HTML object. This property could have been specified in the corresponding declaration.

The "SELECT_FORM.SAILBOARDS.CHECK" declaration contains property information. The "value" attribute points to a sailboard object that contains the name of the first/next sailboard to populate the repeating subcomponent in the HTML document. The return attribute specifies the variable, "selectionArray" into which an input value is pushed when the HTML document returns from the client. The HTML object that is instantiated from the "SELECT_FORM.SAILBOARDS.CHECK" declaration generates an INPUT HTML statement of type "checkbox" when it is receives the "create" message.

The "SELECT_FORM.SAILBOARDS.NAME" declaration file entry identifies the NSWString object class. An NSWString object is instantiated and inserted into the SELECT_FORM.SAILBOARDS" nested group. The "NSWString" object generates a string that is equal to the value of the first/next sailboard.

The NSWTAG extension can be replaced using data stored in properties of the object(s) associated with the HTML element and NSWTAG extension. Table Three provides an example of HTML including NSWTAG extensions contained within an ordered list HTML element. The NSWTAG extensions provide associations to entries in the declarations file that point to data stored in object properties.

TABLE THREE

```
<HTML>
    <NSWTAG NAME="SELECT_FORM">
        <FORM>
                        . . .
            <OL>
            <NSWTAG NAME="WEBPEOPLE">
                <NSWTAG NAME="FN"><INPUT TYPE="text">
                  </NSWTAG>
                <NSWTAG NAME="LN"><INPUT TYPE="text">
                  </NSWTAG>
                <P>
            </NSWTAG>
            </OL>
                        . . .
        </FORM>
    </NSWTAG
                        . . .
</HTML>
```

A "WEBPEOPLE" NSWTAG extension is contained within the "SELECT_FORM" NSWTAG extension. The "WEBPEOPLE" NSWTAG extension includes two nested group HTML elements. Associated with each HTML group elements is a corresponding declaration file entry. The following provides an example of declaration file entries for the "WEBPEOPLE", "FN", and "LN" NSWTAG extensions, respectively:

SELECT_FORM.WEBPEOPLE:NSWRepetition {list=web_dept; item=employee};

SELECT_FORM.WEBPEOPLE.FN:NSWInput {value=employee fn},

SELECT_FORM.WEBPEOPLE.LN:NSWInput {value=employee.ln};

The "WEBPEOPLE" declaration file entry indicates that the "WEBPEOPLE" NSWTAG extension is a repeating subcomponent or group of HTML. Group objects are instan-

tiated for each. When the HTML document is rendered, the "WEBPEOPLE" group object sends a "create" message to the "FN" and "LN" group objects. The "FN" and "LN" group objects send a "create" message to the two "NSWInput" objects instantiated as a result of the "SELECT_FORM.WEBPEOPLE.FN" and "SELECT_FORM.WEBPEOPLE.LN" declarations. The declarations specify the data associated with the "fn" and "In" properties of an employee object. The "NSWInput" objects generate HTML statements in the HTML document using the values of the "fn" and "In" properties of the employee object.

The "FN" and "LN" groups are contained within the "WEBPEOPLE" group. The "WEBPEOPLE" group is a repetition group. Therefore, "FN" and "LN"groups can be activated multiple times to generate HTML statement blocks within the HTML document.

The name property of the group, or NSWTAG, extension provides the ability to identify the group and the statements contained within the group. During parsing, a name property associated with a group can be stored in a hash table. During processing, an object can be located by traversing the object tree. Preferably, however, one object can retrieve group names from the hash table to locate another object within the object tree. FIG. **6**B modifies the flow provided in FIG. **6**A to include steps to add tag name information to a hash table.

Steps **614** and **616** are added to the flow. After the object is added to the object tree at step **612**, a determination is made whether the parsed statement contained a name property. If there is no name property, processing continues at step **602** to process any remaining elements of the HTML template. If a name property is defined, processing continues at step **616** to store the identifier in the hash table. Processing continues at step **602** to process any remaining elements of the HTML template.

Thus, a method and apparatus for generating object-oriented world wide web pages is provided.

APPENDIX A

Class	Description
NSWHAnchor	Object cover for Anchor HTML element
NSWHCheckbox	Object cover for Input HTML element, type is "checkbox"
NSWHComment	Object cover for comments
NSWHElement	
NSWHForm	Object cover for Form HTML element
NSWHGroup	Cover for group extension. A group is a container for other objects.
NSWHImage	Object cover for Image HTML element
NSWHImageInput	Object cover for Input HTML element, type is "image"
NSWHImagemapRegion	
NSWHInput	Abstract class for input HTML elements. All input HTML elements subclass this class
NSWHKeyValueCoding	Defines methods to maintain a dictionary of data values associated with keys. Values can be extracted from the dictionary to initialize an object, or retrieved from an object and stored in the dictionary.
NSWHList	A generic container class for HTML elements that can contain other elements or plain text. This class can be used for HTML elements that do not covered by another object class
NSWHListItem	Object cover for List Item HTML element
NSWHMarkedElement	
NSWHMenu	Object cover for Menu HTML element
NSWHOption	Object cover for Option HTML element
NSWHOrderedList	Object cover for Ordered List HTML element
NSWHPage	
NSWHRadio	Object cover for Input HTML element, type is "radio"
NSWHRectRegion	

APPENDIX A-continued

Class	Description
NSWHRepetitionGroup	Object cover for repeating group HTML extension
NSWHReset	Object cover for Input HTML element, type is "reset"
NSWHSelect	Object cover for Select HTML element
NSWHString	Object cover for free-floating HTML text
NSWHSubmit	Object cover for Input HTML element, type is "submit"
NSWHText	Object cover for Input HTML element, type is "text"
NSWHTextArea	Object cover for TextArea HTML element
NSWHParser	
NSWHUnknownElement	Object cover for Unordered List HTML element
NSWHUnorderedList	Object cover for Input HTML element
NSWHUrl	Object cover for Uniform Resource Locator
NSWHUrn	Object cover for URN

What is claimed is:

1. A method of dynamically generating Hypertext Markup Language (HTML) elements using objects in a computer system, comprising the steps of:

defining object classes for each of said elements of said HTML;

creating an HTML template, said HTML template having a statement that defines one of said HTML elements;

identifying the object class associated with said one of said HTML elements;

creating an instance of said object class, said instance containing data and methods for processing said one of said HTML elements; and

transmitting a "create" message to said instance to dynamically generate a plurality of HTML statements using the data and methods of said instance.

2. The method of claim **1** wherein the step of identifying comprises the steps of:

creating a correspondence between each of said HTML elements and said object classes;

parsing said statement in said HTML template to identify said one of said HTML elements defined by said statement;

identifying, using said correspondence, the object class associated with said one of said HTML elements.

3. The method of claim **2** wherein said step of creating said correspondence further comprises the step of naming an object class the same name as its corresponding HTML element.

4. The method of claim **2** wherein said step of creating said correspondence further comprises the step of creating a table that contains a plurality of entries, one of said plurality entries contains the name of an HTML element and the name of its corresponding object class.

5. The method of claim **1** further comprising the steps of:

identifying the object class associated with a second HTML element;

creating a second instance of said object class associated with said second HTML element, said second instance containing data and methods for processing said second of said HTML elements;

transmitting a message to said second instance to execute a plurality of procedures that process information received in a client request using the data and methods of said second instance.

6. The method of claim **5** wherein said information is used to update an external data source.

7. The method of claim 1 further comprising the steps of:

identifying the object class associated with a second HTML element;

creating a second instance of said object class associated with said second HTML element, said second instance containing data and methods for processing said second of said HTML elements;

transmitting a message to said second instance to execute a plurality of procedures to modify the properties of said instance using the data and methods of said second instance.

8. The method of claim 7 wherein said plurality of procedures modify the properties of said instance using data retrieved from an external data source.

9. A method of dynamically generating a Web page in a computer system comprising the steps of:

identifying a plurality of object instances corresponding to a plurality of Web Page definitional statements;

generating said plurality of Web page definitional statements using said instances where such generation may be based on structure determined at runtime.

10. The method of claim 9 further comprising the step of:

creating an object tree, said object tree defining a hierarchy for said instances, said hierarchy having a root level and a plurality of descendent levels.

11. The method of claim 10 wherein said generating step further comprises the steps of:

activating said root level of said object tree;

transmitting a message traversing said object tree to each of said plurality of descendent levels to generate said plurality of Web page definitional statements.

12. The method of claim 10 wherein said generating step further comprises the step of:

executing a plurality of procedures that make a plurality of modifications to said object tree.

13. The method of claim 12 wherein said plurality of modifications include the insertion of a plurality of object instances in said object tree.

14. The method of claim 12 wherein said plurality of modifications include the modification of properties of said object instances.

15. The method of claim 9 further comprising processing a plurality of elements of said Web page as a group, said processing comprising the steps of:

creating a group element;

creating element objects for each of said plurality of Web page elements and a group object class for said group element;

specifying that said plurality of elements are contained within said group element;

transmitting a message to be performed by said group to an instance of said group object class;

said group object instance forwarding said message to each of said element objects in said group;

said plurality of element objects processing said message.

16. The method of claim 15 wherein said step of creating said group object further comprises the step of:

initializing properties of said group object and said element objects using a plurality of property definitions contained in a declaration, wherein said declaration contains definitional information;

adding a plurality of new element objects to said group using a plurality of object definitions contained in said declaration.

17. The method of claim 9 further comprising processing a plurality of elements that recur a plurality of times within said Web page as a group, said processing comprising the steps of:

creating an object for said group;

creating element objects for each of said plurality of elements;

specifying that said plurality of elements are contained within said group;

transmitting a message to performing a method of an instance of said group;

said group instance forwarding said message to performing methods of said plurality of element objects a plurality of times;

said element objects processing said message performing said methods each of said plurality of times.

18. The method of claim 17 wherein said plurality of element objects generate a block of definitional statements for said Web page each of said plurality of times.

19. The method of claim 18 further comprising the steps of:

retrieving data from an external source;

generating said block of definitional statements using said data.

20. The method of claim 19 wherein said external source is a database management system.

21. The method of claim 19 wherein said step of retrieving further comprises the steps of:

creating an object, said object having a plurality of properties and methods;

retrieving said data from said external source into said plurality of properties of said object.

22. An article of manufacturing comprising:

a computer usable medium having computer readable program code embodied therein for generating a Web page comprising:

computer readable program code configured to cause a computer to instantiate a page object;

computer readable program code configured to cause a computer to identify a component of an HTML definitional statement;

computer readable program code configured to cause a computer to instantiate a component object to process said component of said HTML definitional statement; and

computer readable program code configured to cause a computer to generate said Web page using said page object and said component object.

23. The article of manufacture of claim 22 further comprising:

computer readable program code configured to cause a computer to create an object tree with said page object as the root of said object tree;

computer readable program code configured to cause a computer to insert said component object in said object tree as a child of said page object.

24. The article of manufacture of claim 22 wherein said computer readable code configured to cause a computer to generate said Web page further comprising:

computer readable program code configured to cause a computer to activate said page object;

computer readable program code configured to cause a computer to generate a first plurality of definitional statements for said Web page;

computer readable program code configured to cause a computer to activate said component object;

computer readable program code configured to cause a computer to generate a second plurality of definitional statements for said Web page.

25. A method of processing a client request to generate a Web page in a computer system comprising the steps of:

receiving said client request to generate said Web page;

instantiating said objects to satisfy said request, whereas said objects comprise HTML definitional statements;

processing said request for said Web page using said objects.

26. The method of claim **25** wherein said objects are Web page element objects, said method further comprising the steps of:

said element objects generating said Web page;

sending said Web page to said client.

27. The method of claim **26** further comprising the steps of:

receiving said Web page from said client;

said element objects processing information stored in said Web page.

28. A method for programmatically manipulating HTML definitional statements in a computer system comprising:

utilizing a one-to-one mapping between elements of HTML and object classes to generate an object tree wherein a node object of said object tree represents at least one HTML definitional statement;

generating a Web page dynamically utilizing said object tree by instantiating each object in said object tree in hierarchical order, wherein said instantiation generates said at least one HTML definitional statement.

29. A method for dynamically displaying an electronic mail message on a Web page in a computer system comprising:

fetching said electronic mail messages to display;

creating a first object that corresponds to at least one HTML element;

creating a second object representing said electronic mail message comprising a sender name, subject, and message content;

creating an object tree wherein said first object is a parent to said second object;

dynamically generating said Web page from said object tree displaying said electronic mail message.

* * * * *

Exhibit 46

US006249291B1

(12) **United States Patent**
Popp et al.

(10) **Patent No.: US 6,249,291 B1**
(45) **Date of Patent: *Jun. 19, 2001**

(54) **METHOD AND APPARATUS FOR MANAGING INTERNET TRANSACTIONS**

(75) Inventors: **Nicolas Popp**, Menlo Park; **Bruce Ong**; **Charles D'Harcourt**, both of San Francisco, all of CA (US)

(73) Assignee: **NeXT Software, Inc.**, Redwood City, CA (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/532,491**

(22) Filed: **Sep. 22, 1995**

(51) **Int. Cl.**[7] **G06T 9/00**
(52) **U.S. Cl.** **345/473**
(58) **Field of Search** 395/761, 326–332; 707/2; 345/473

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,297,249	*	3/1994	Bernstein et al.	395/329
5,600,831	*	2/1997	Levy et al.	707/2
5,623,656	*	4/1997	Lyons	395/610

OTHER PUBLICATIONS

Electronic Mail Message: "Pricing Model—WebRex—TRUTH wanted!." Sep. 10, 1995, Ted Shelton, Information Technology Solutions, Inc.

Documents Posted on World Wide Web: "The WadsWorth Concept," last modified Aug. 10, 1995.

Documents Posted on World Wide Web: "Building Client/Server/Web Applictaions, How Fast Can Your Business Respond?", retrieved May 6, 1997.

Documents Posted on World Wide Web: "The Object–Oriented Development Environment For Creating Dynamic Web Applications," Visual Wave 2.0, retrieved May 5, 1997.

Aronson "HTML3 Manual of Style" pp 44–45, 112–123, 1995.*

"Introduction To Visual Basic" by Application Developers Training Co. pp 188–189, 1994.*

Aronson "HTML 3 Manual of Style" pp 111–123, 1995.*

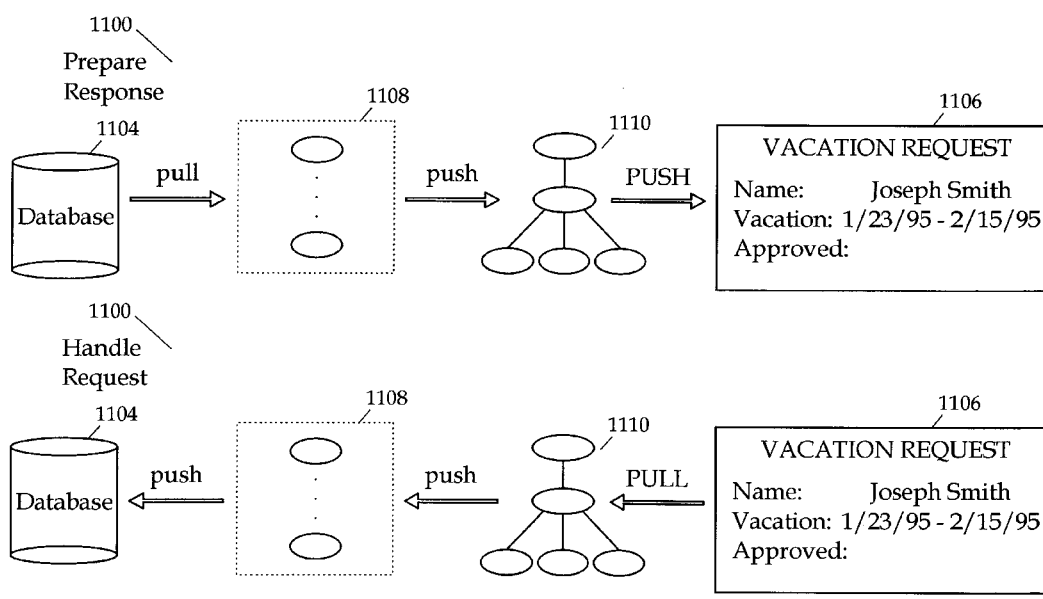
* cited by examiner

Primary Examiner—Almis R. Jankus
(74) *Attorney, Agent, or Firm*—The Hecker Law Group

(57) **ABSTRACT**

An object-oriented approach provides the ability to develop and manage Internet transactions. Local applications can be accessed using any workstation connected to the Internet regardless of the workstation's configuration. Some or all of a Web page can be generated dynamically using input received in a returned page, generated at runtime, or retrieved from an external data source (e.g., database or electronic mail system). When the Web page definition is rendered, the information contained in template(s), declaration file(s), and object(s) are used to generate standard definitions. State information is maintained across transactions. Using state information, virtual applications, sessions, transactions, and pages can be implemented. Self-contained modules, or components, provide the ability to share implementations and create multi-content documents. Event objects can be used to assist in event handling management.

25 Claims, 22 Drawing Sheets



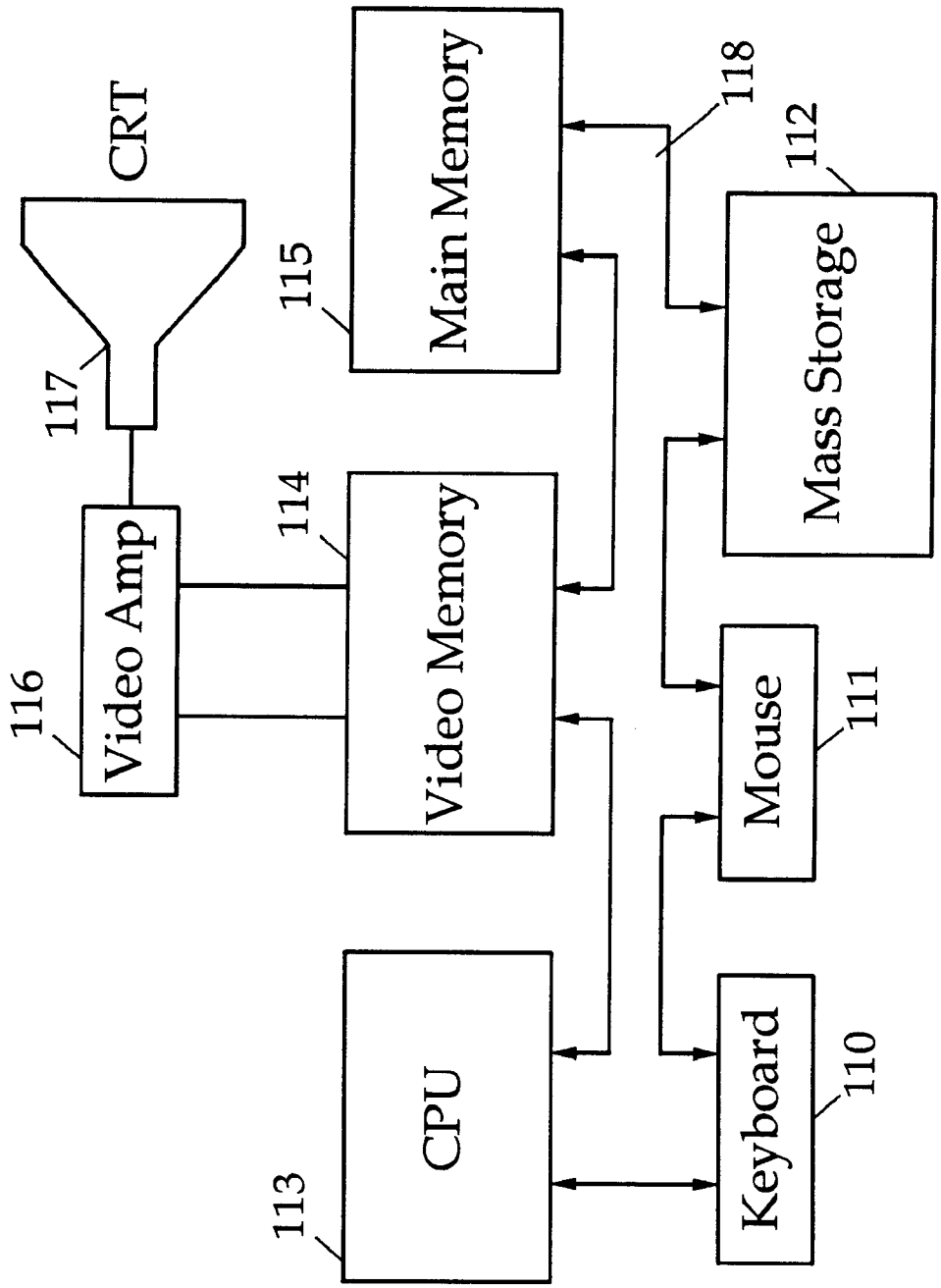

CRT
117
Video Amp
116
Main Memory
115
Video Memory
114
CPU
113
Mass Storage
112
118
Mouse
111
Keyboard
110

Figure 1

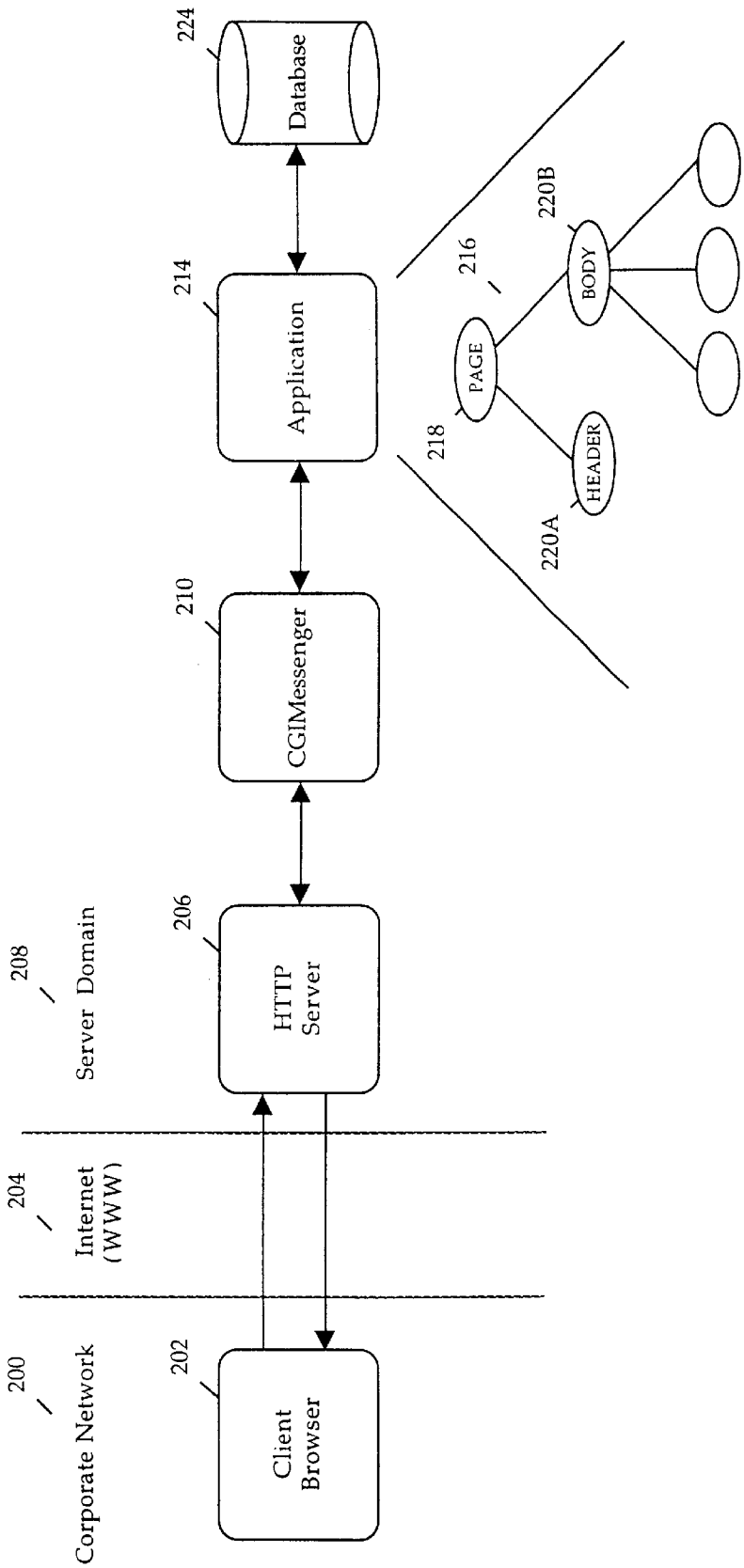

224
Database
214
Application
210
CGIMessenger
206
HTTP Server
208
Server Domain
204
Internet (WWW)
200
Corporate Network
202
Client Browser
216
220B
BODY
PAGE
218
HEADER
220A

Figure 2

MAC003931

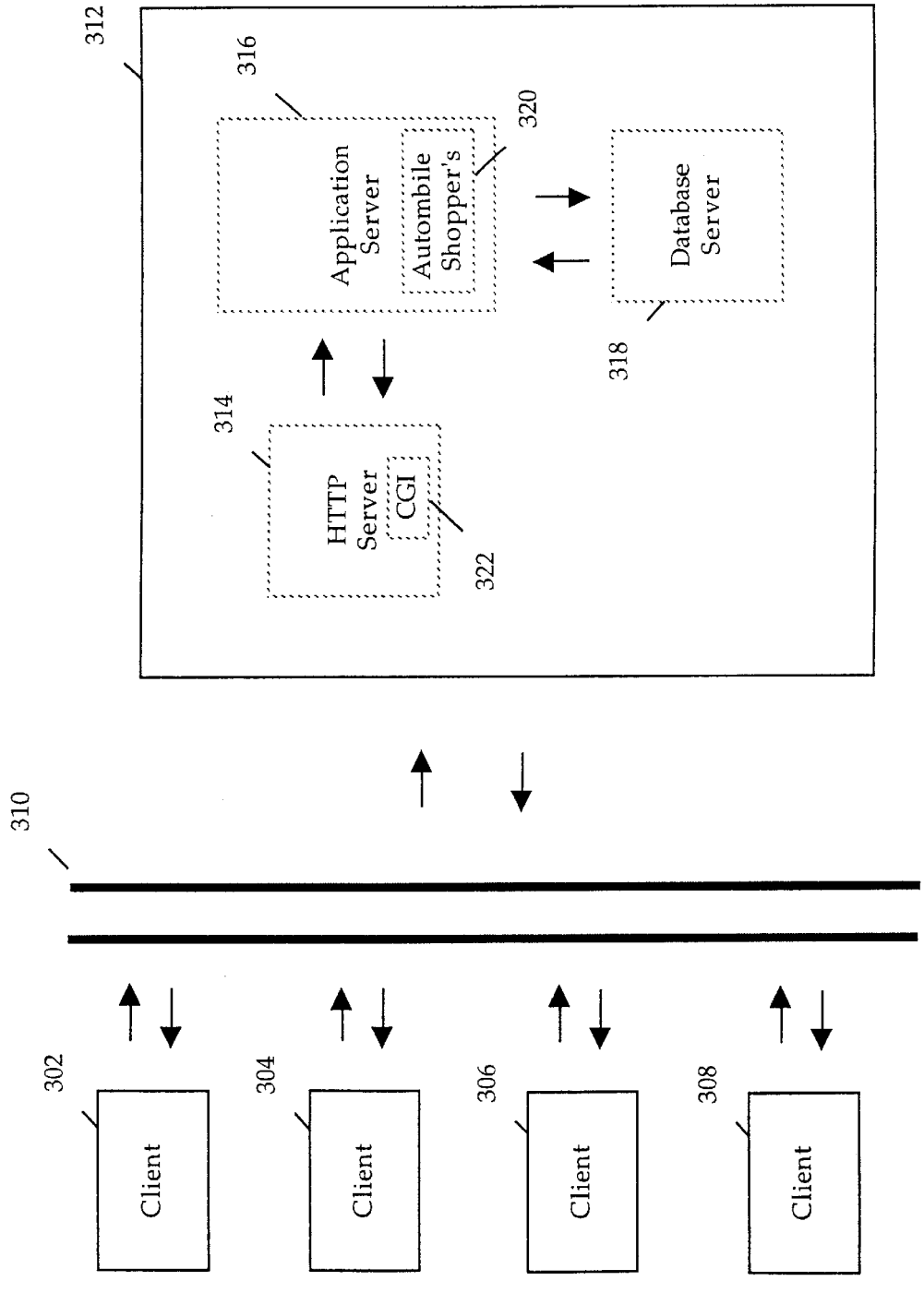


Figure 3A

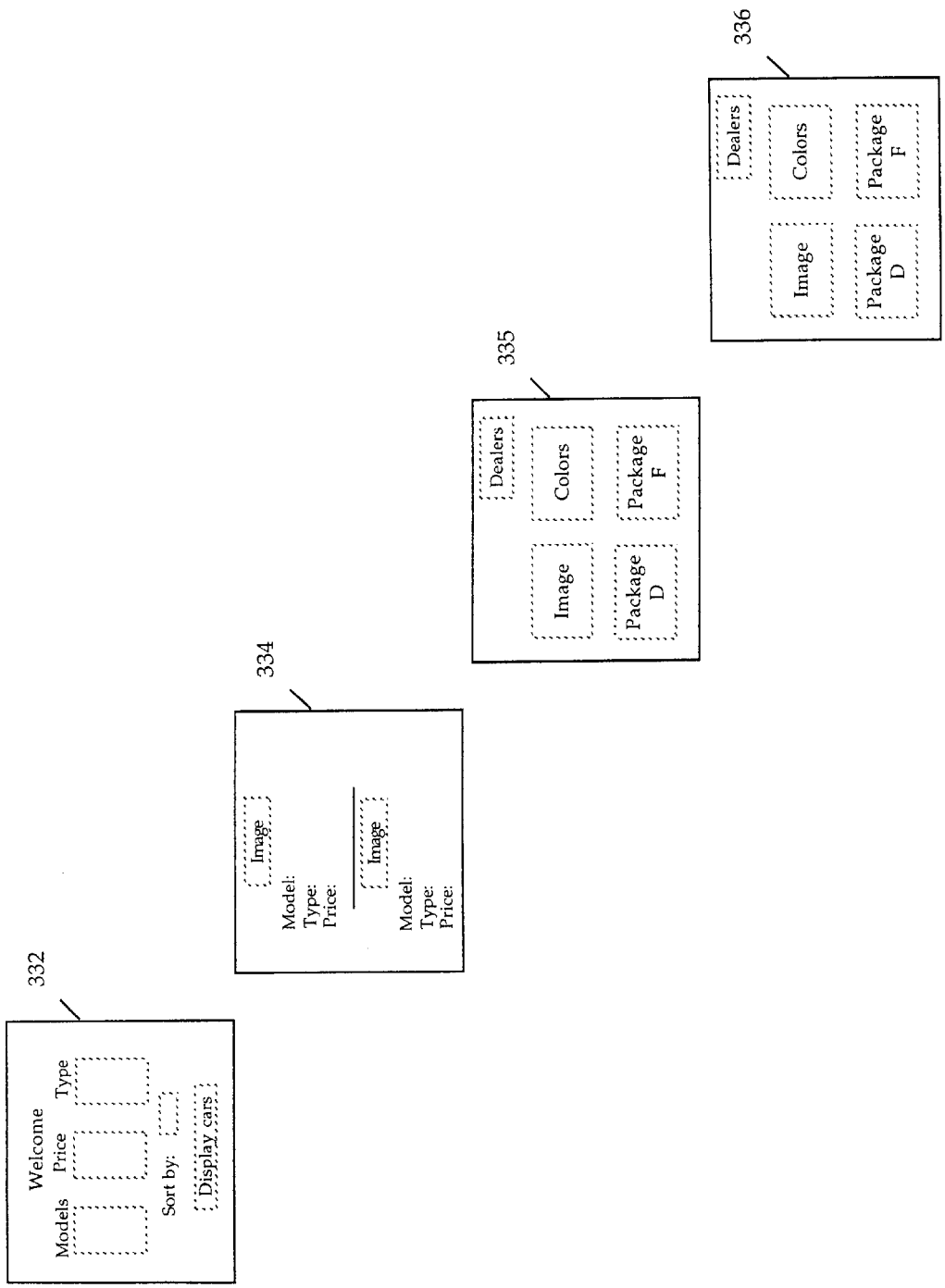

332
Welcome
Models
Price
Type
Sort by:
Display cars
334
Image
Model:
Type:
Price:
Image
Model:
Type:
Price:
335
Dealers
Colors
Image
Package D
Package F
336
Dealers
Colors
Image
Package D
Package F

Figure 3B

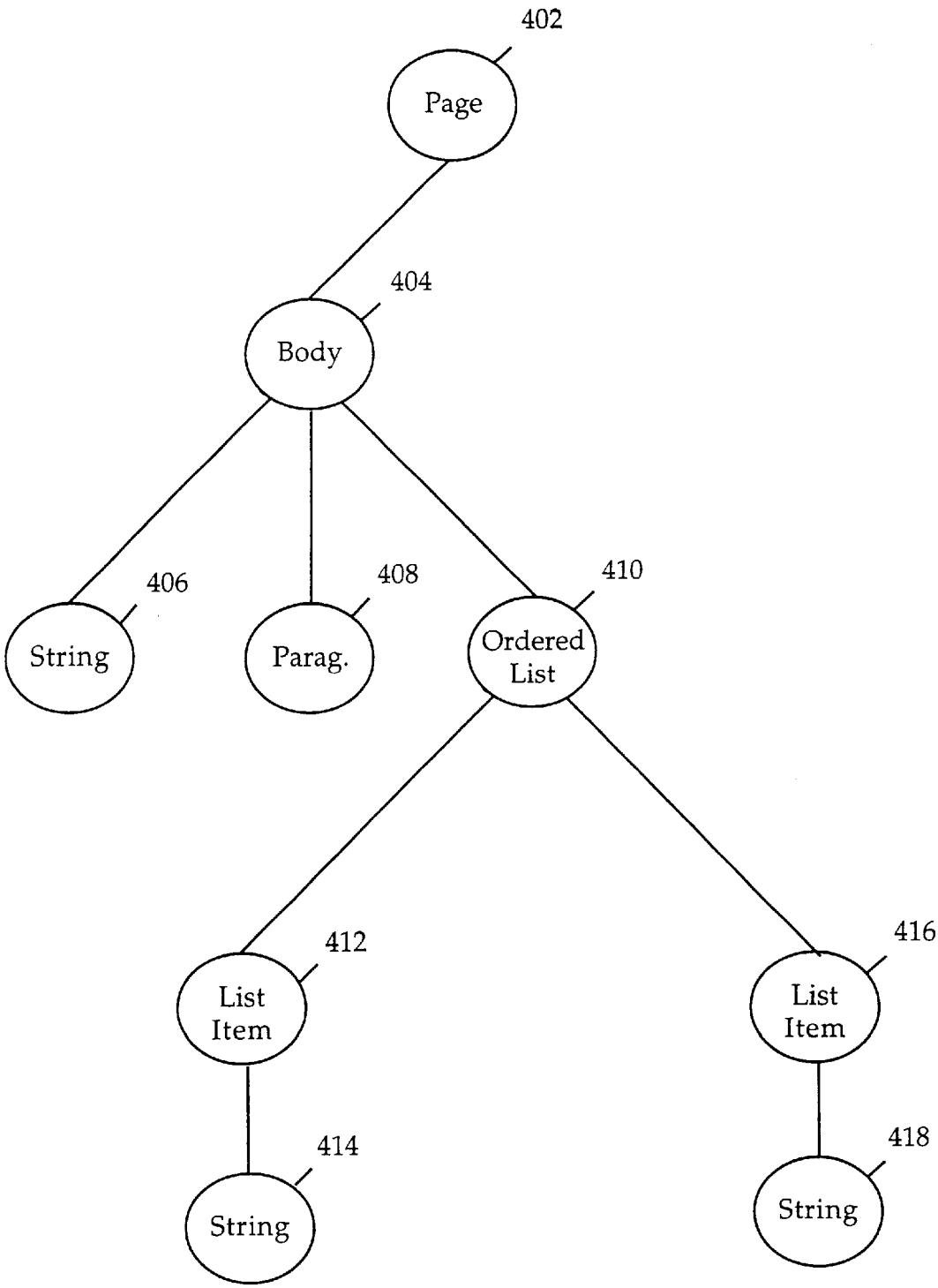

402
Page
404
Body
406
String
408
Parag.
410
Ordered List
412
List Item
414
String
416
List Item
418
String

Figure 4

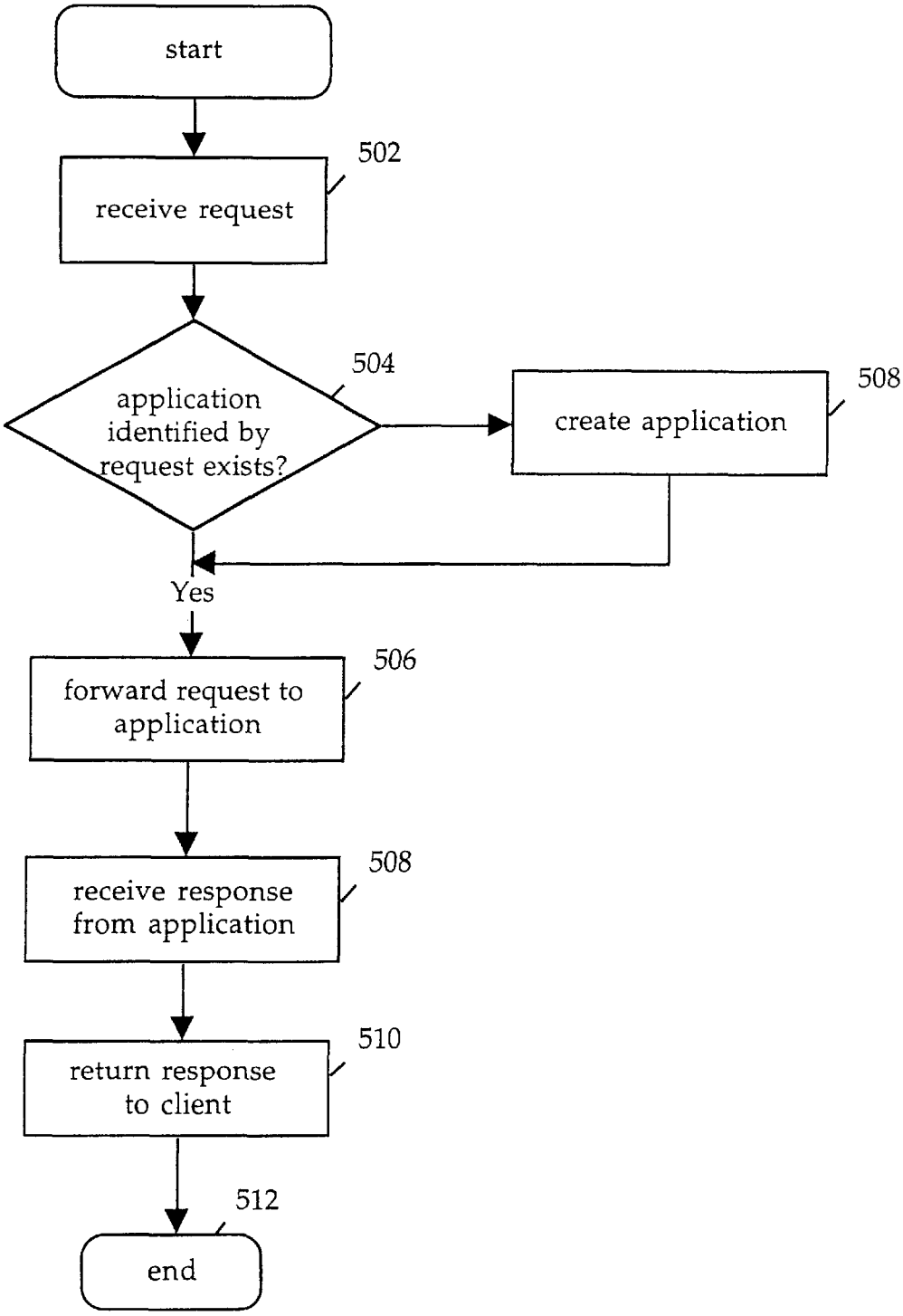

start
receive request
502
application identified by request exists?
504
create application
508
Yes
forward request to application
506
receive response from application
508
return response to client
510
end
512

Figure 5A

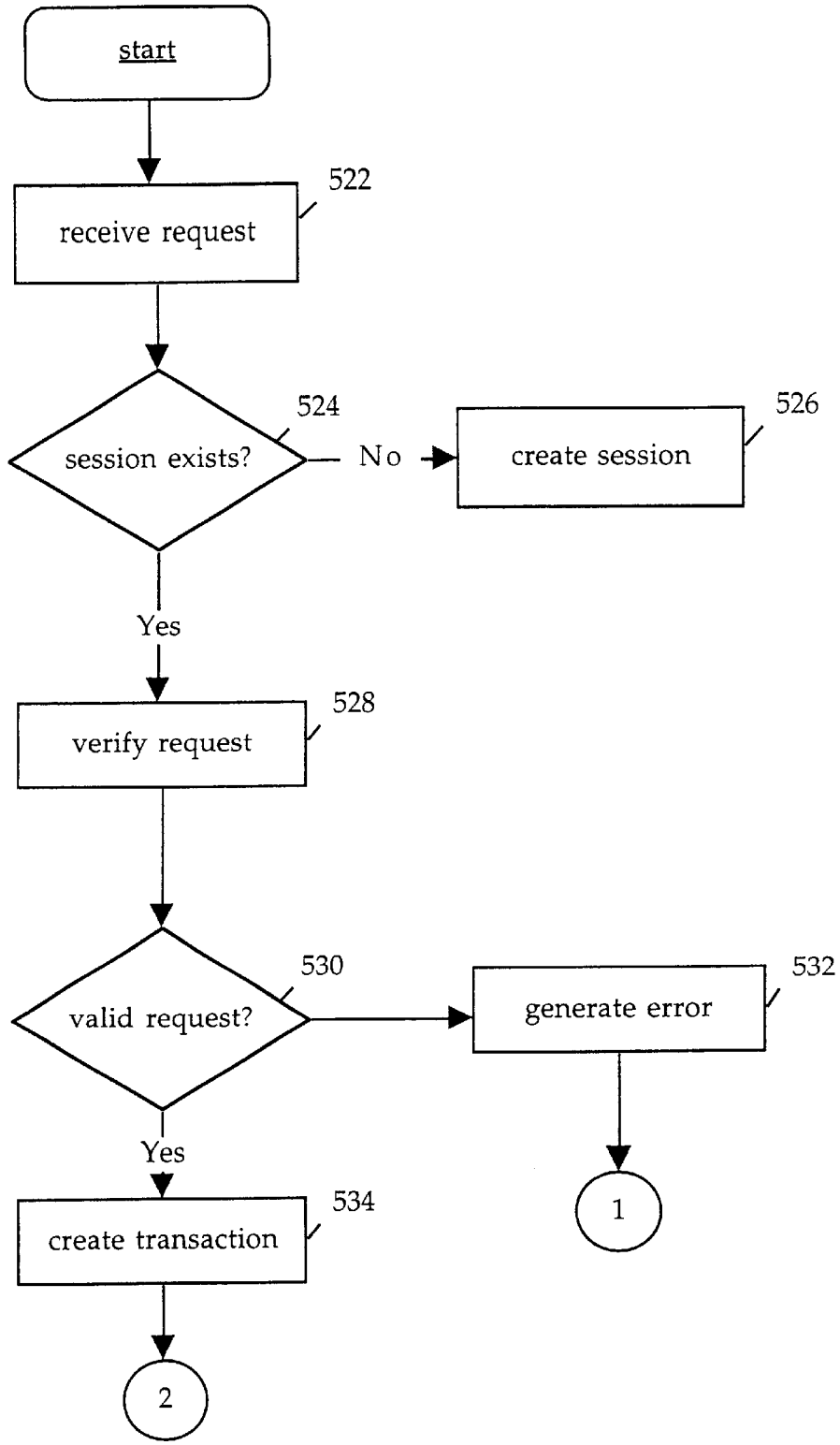

start
receive request
522
session exists?
524
No
create session
526
Yes
verify request
528
valid request?
530
generate error
532
Yes
1
create transaction
534
2

Figure 5B

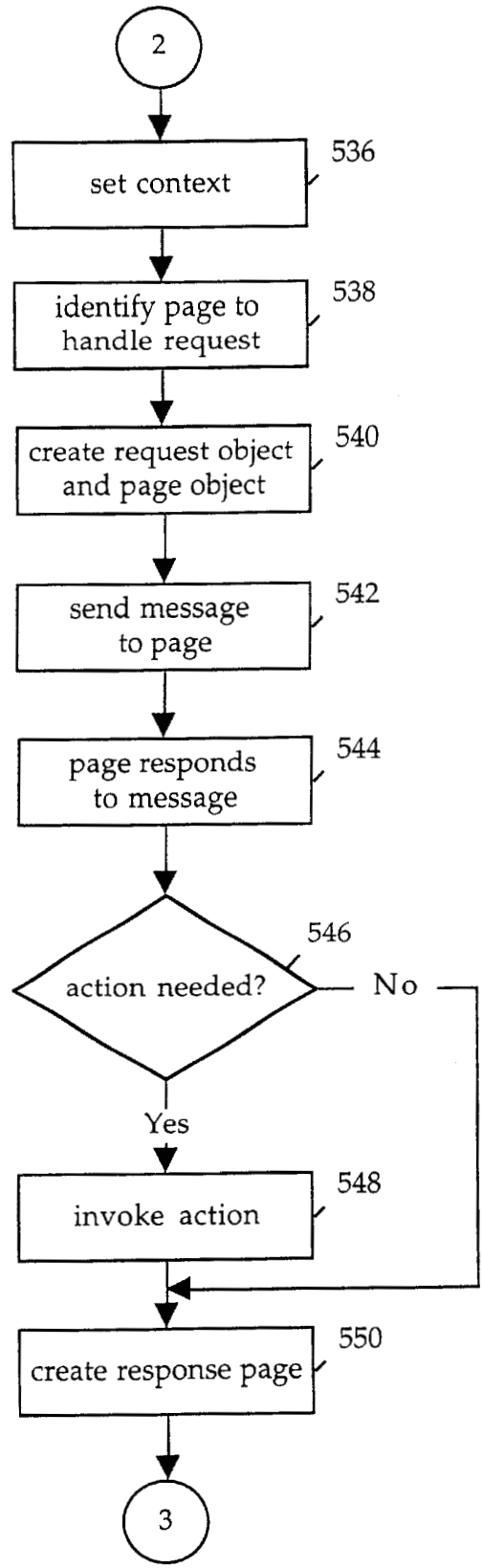

2
set context
536
identify page to handle request
538
create request object and page object
540
send message to page
542
page responds to message
544
action needed?
546
No
Yes
invoke action
548
create response page
550
3

Figure 5C

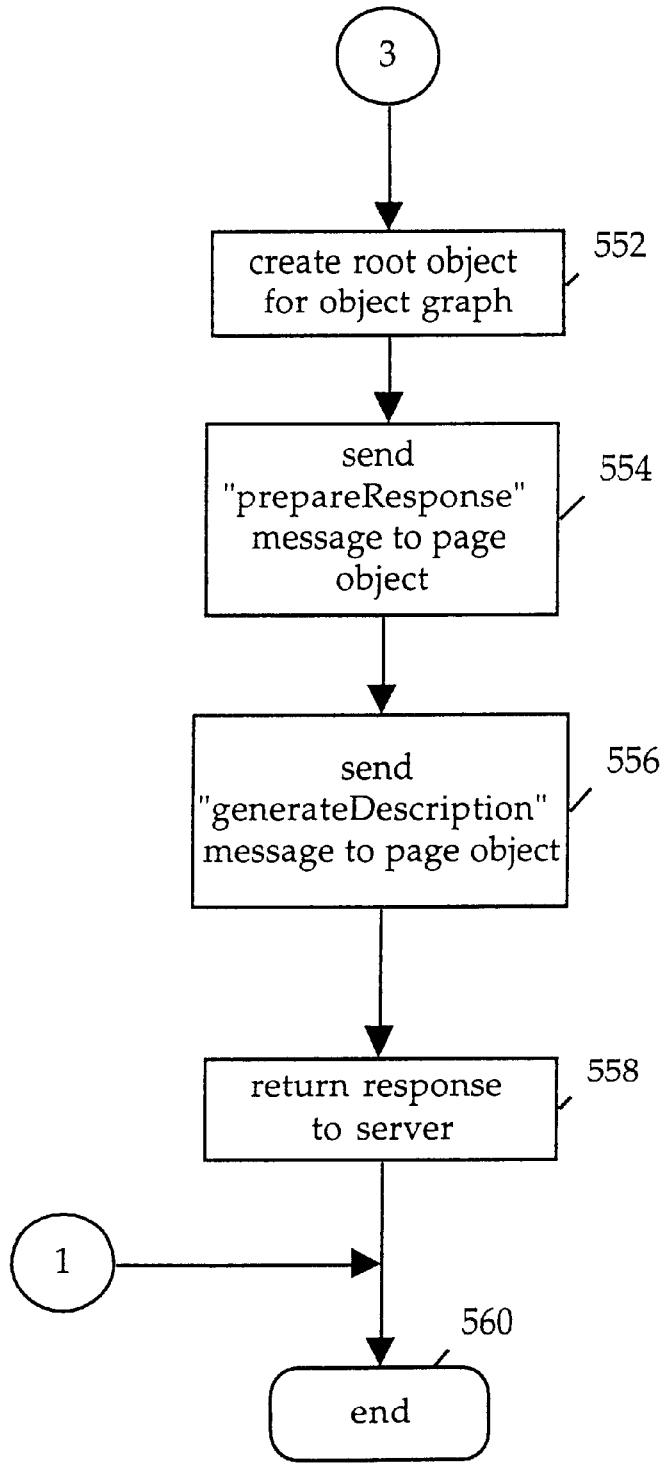

3
create root object for object graph
552
send "prepareResponse" message to page object
554
send "generateDescription" message to page object
556
return response to server
558
1
560
end

Figure 5D

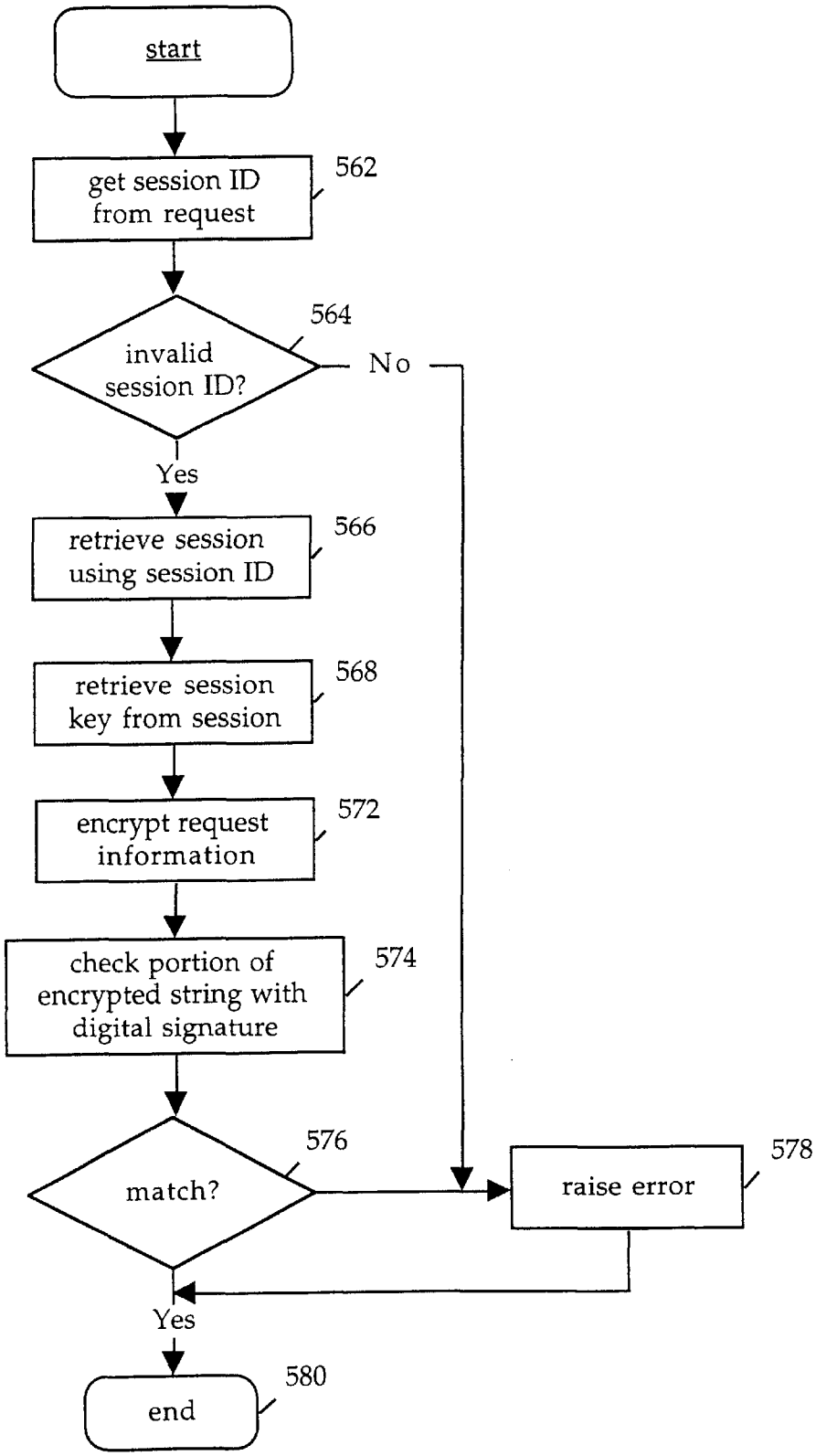

start
get session ID from request
562
invalid session ID?
564
No
Yes
retrieve session using session ID
566
retrieve session key from session
568
encrypt request information
572
check portion of encrypted string with digital signature
574
match?
576
raise error
578
Yes
end
580

Figure 5E

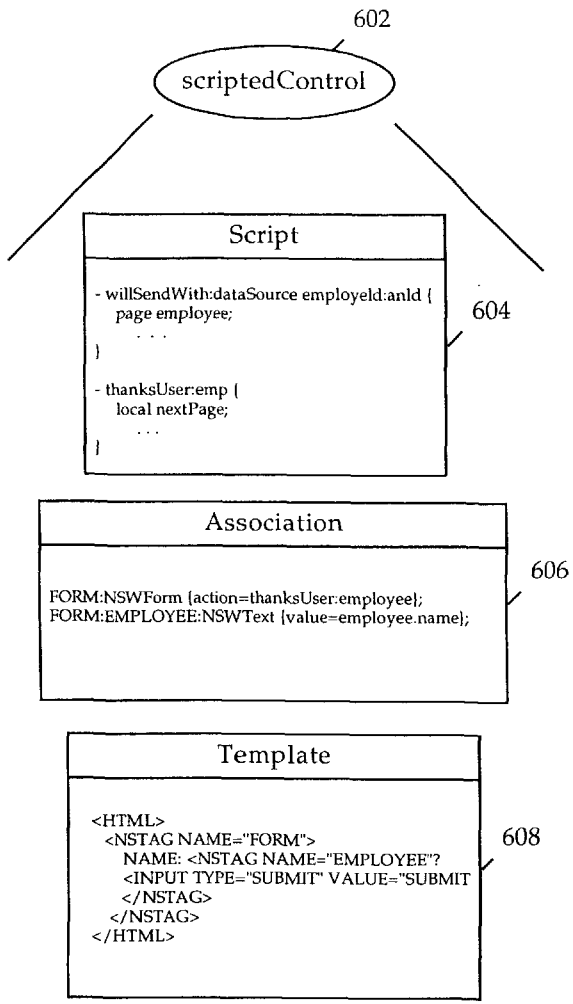

602
scriptedControl
Script
- willSendWith:dataSource employeId:anId {
page employee;
. . .
}
- thanksUser:emp {
local nextPage;
. . .
}
604
Association
FORM:NSWForm {action=thanksUser:employee};
FORM:EMPLOYEE:NSWText {value=employee.name};
606
Template
<HTML>
<NSTAG NAME="FORM">
NAME: <NSTAG NAME="EMPLOYEE"?
<INPUT TYPE="SUBMIT" VALUE="SUBMIT
</NSTAG>
</NSTAG>
</HTML>
608

Figure 6A

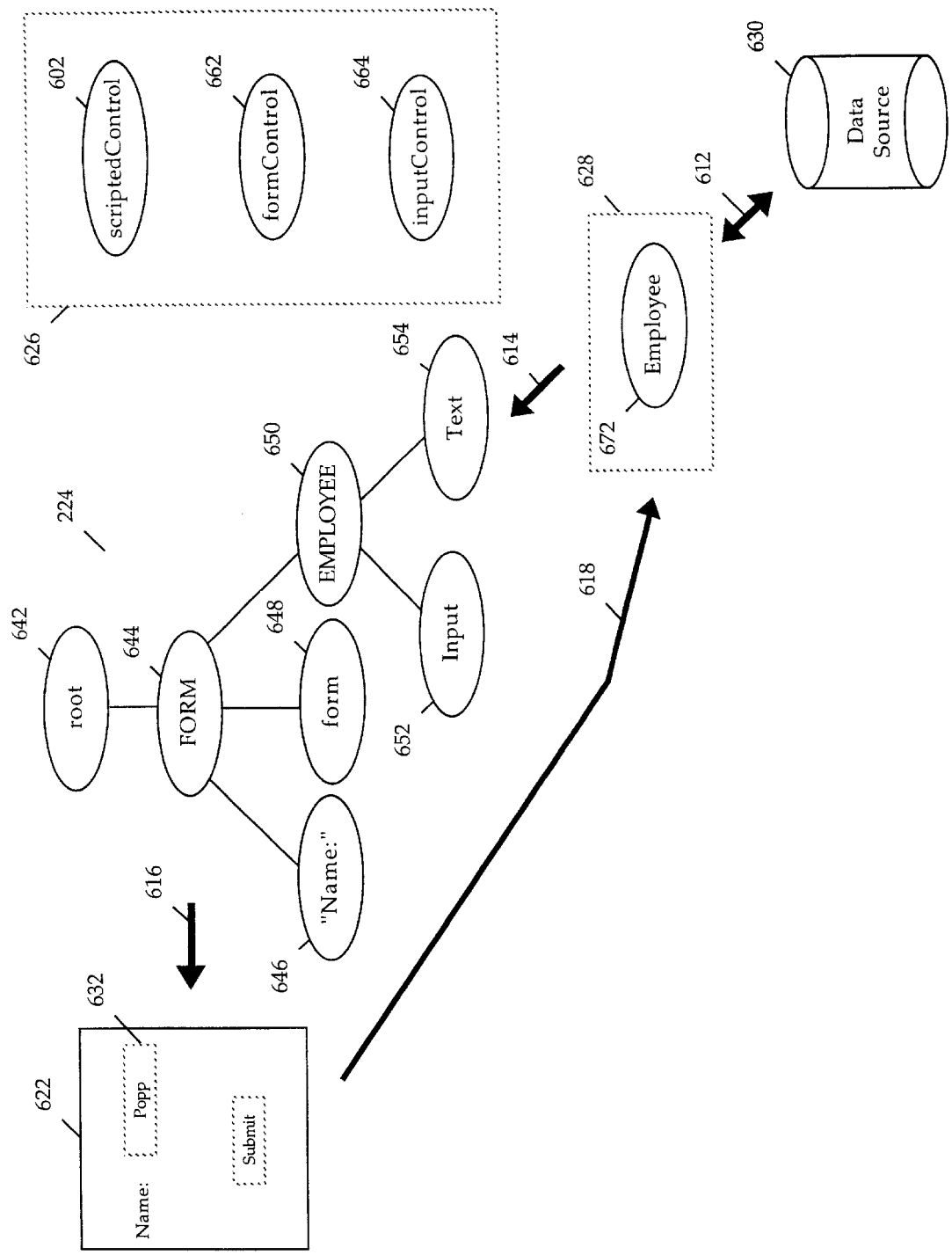

scriptedControl
formControl
inputControl
602
662
664
626
Data Source
630
628
612
Employee
672
614
654
Text
650
EMPLOYEE
224
642
644
root
FORM
648
form
Input
652
618
616
"Name:"
646
632
Popp
Submit
Name:
622

Figure 6B

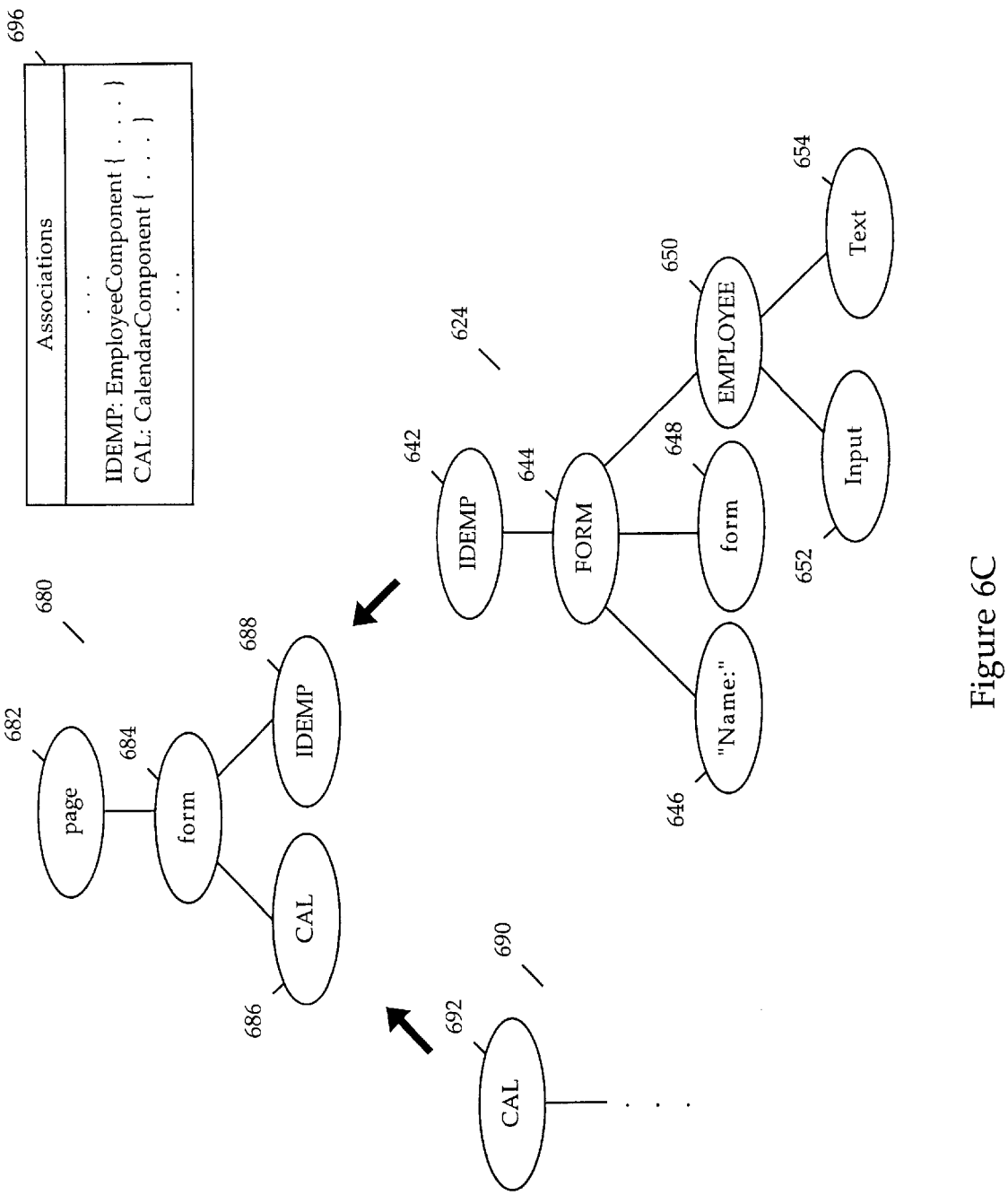

696
Associations
. . .
IDEMP: EmployeeComponent { . . . }
CAL: CalendarComponent { . . . }
. . .
624
642
IDEMP
644
FORM
648
form
646
"Name:"
650
EMPLOYEE
652
Input
654
Text
680
682
page
684
form
688
IDEMP
686
CAL
690
692
CAL
. . .

Figure 6C

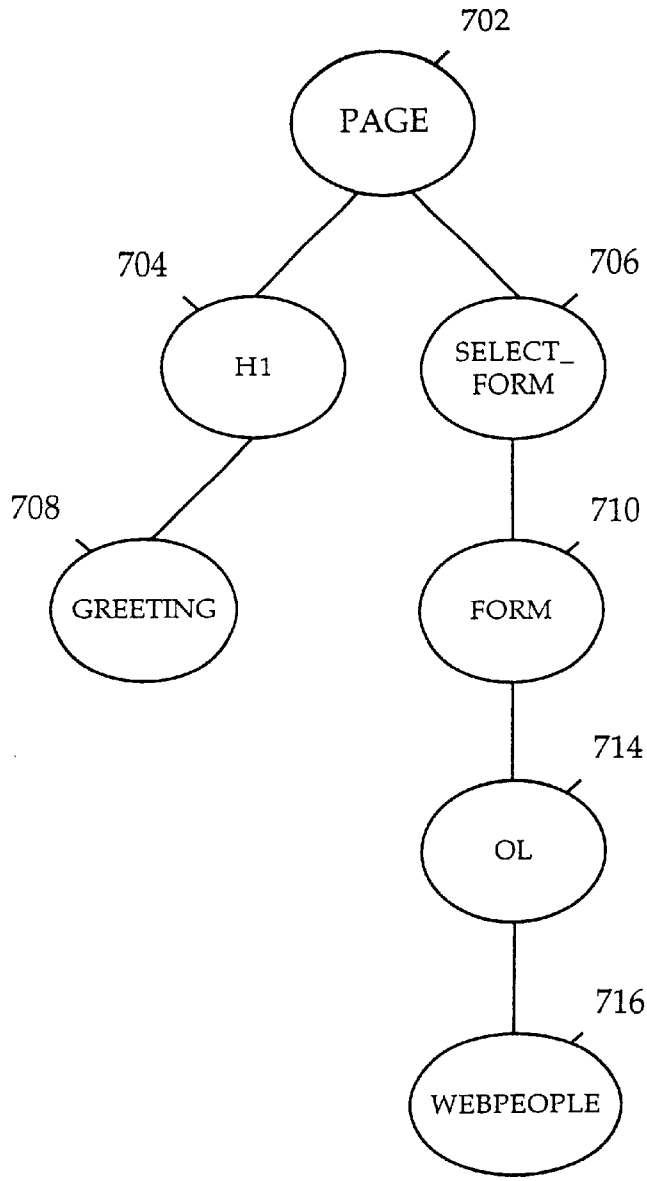

702
PAGE
704
706
H1
SELECT_
FORM
708
710
GREETING
FORM
714
OL
716
WEBPEOPLE

Figure 7A

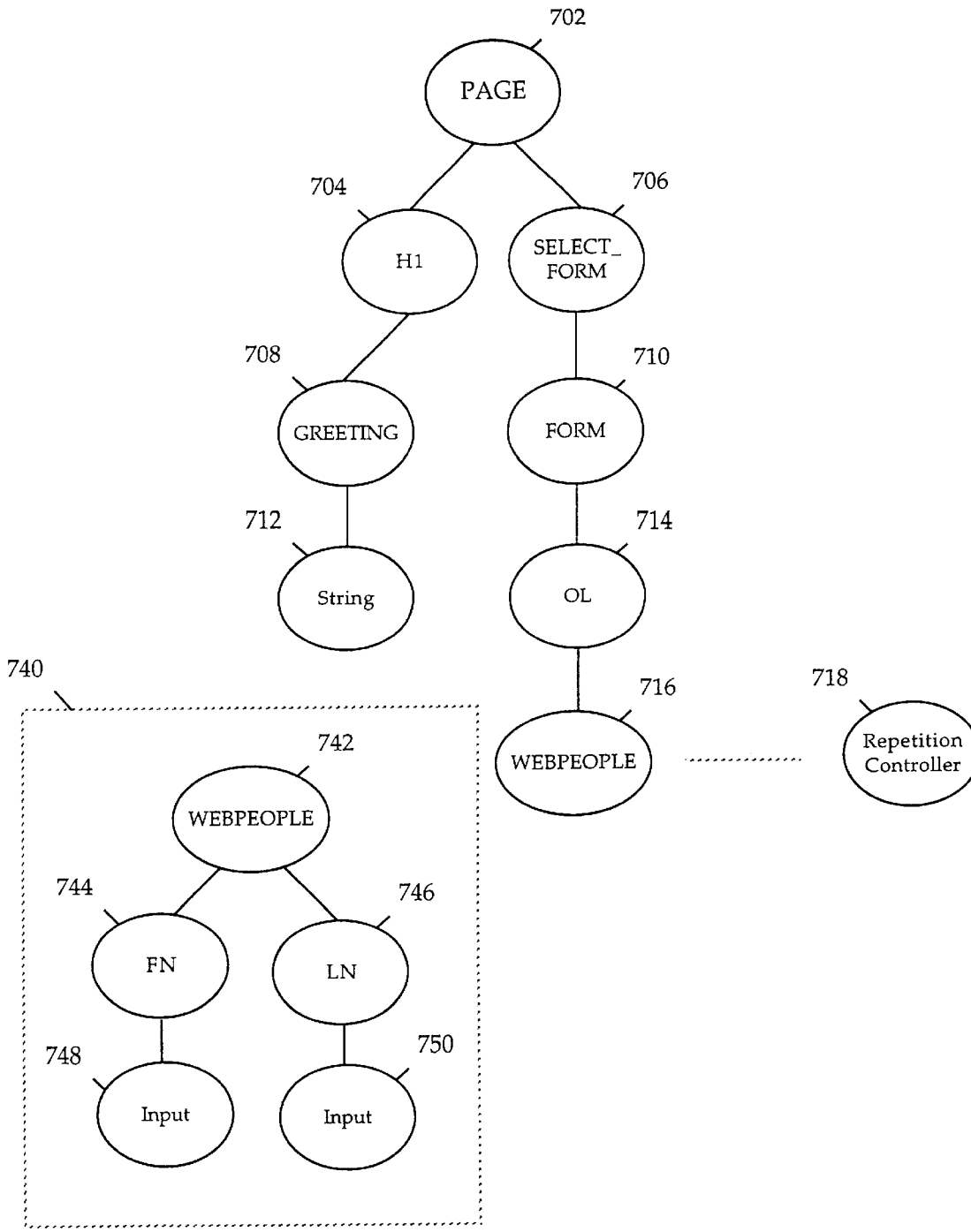

702
PAGE
704
H1
706
SELECT_ FORM
708
GREETING
710
FORM
712
String
714
OL
716
WEBPEOPLE
718
Repetition Controller
740
742
WEBPEOPLE
744
FN
746
LN
748
Input
750
Input

Figure 7B

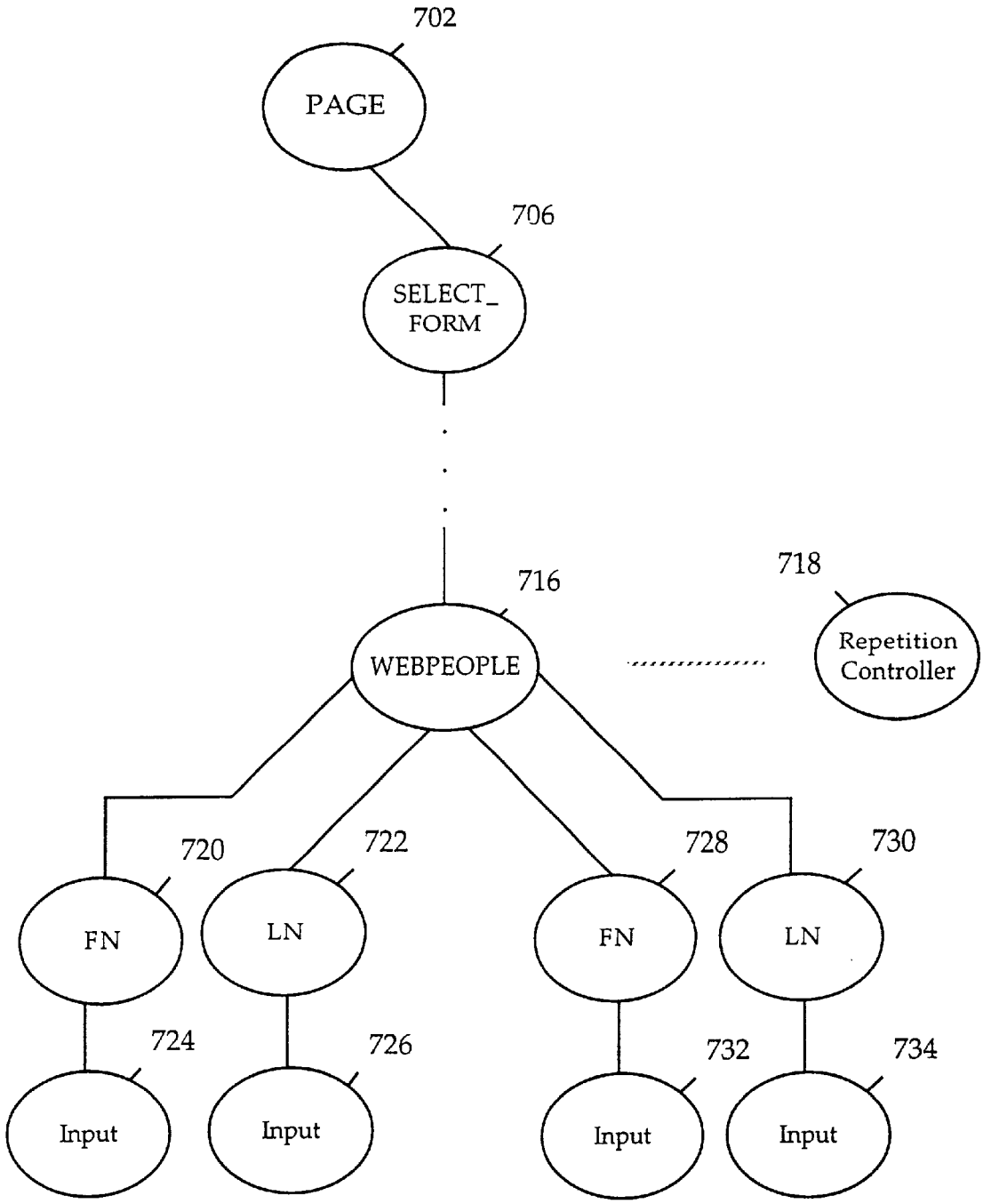

702
PAGE
706
SELECT_
FORM
716
718
WEBPEOPLE
Repetition
Controller
720
722
728
730
FN
LN
FN
LN
724
726
732
734
Input
Input
Input
Input

Figure 7C

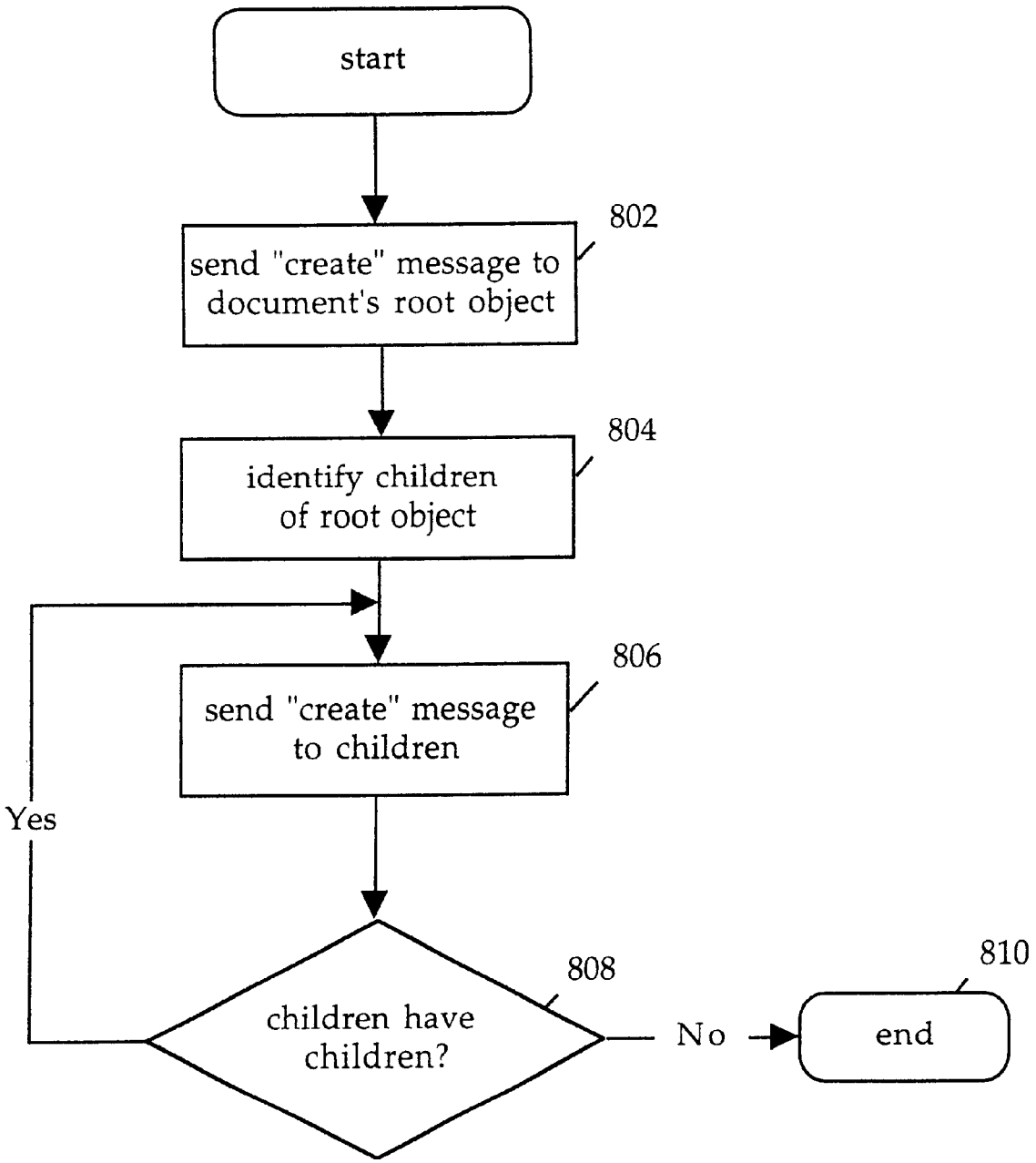

start
send "create" message to document's root object
802
identify children of root object
804
send "create" message to children
806
children have children?
808
Yes
No
end
810

Figure 8

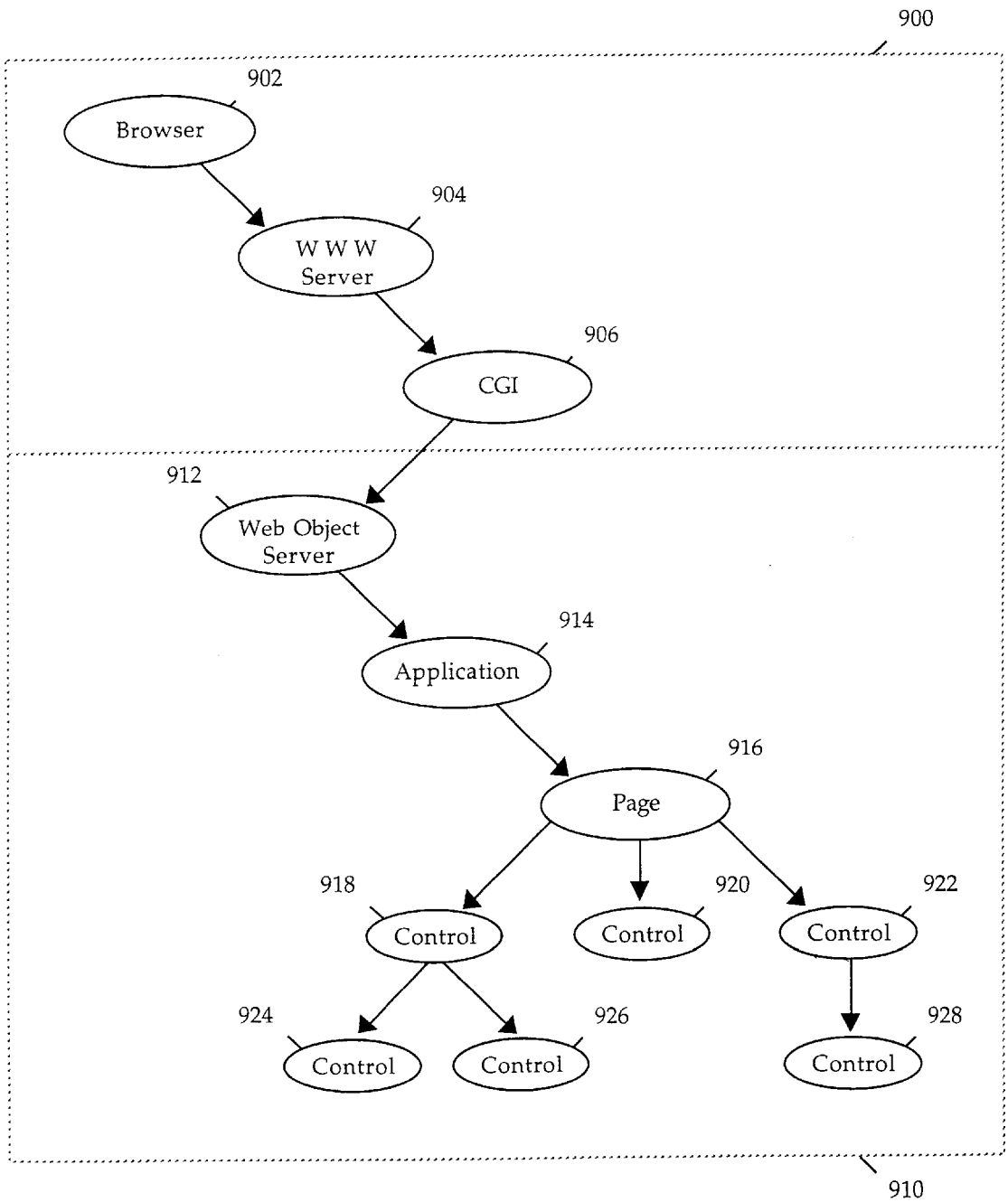

900
902
Browser
904
W W W
Server
906
CGI
912
Web Object
Server
914
Application
916
Page
918
Control
920
Control
922
Control
924
Control
926
Control
928
Control
910

Figure 9A

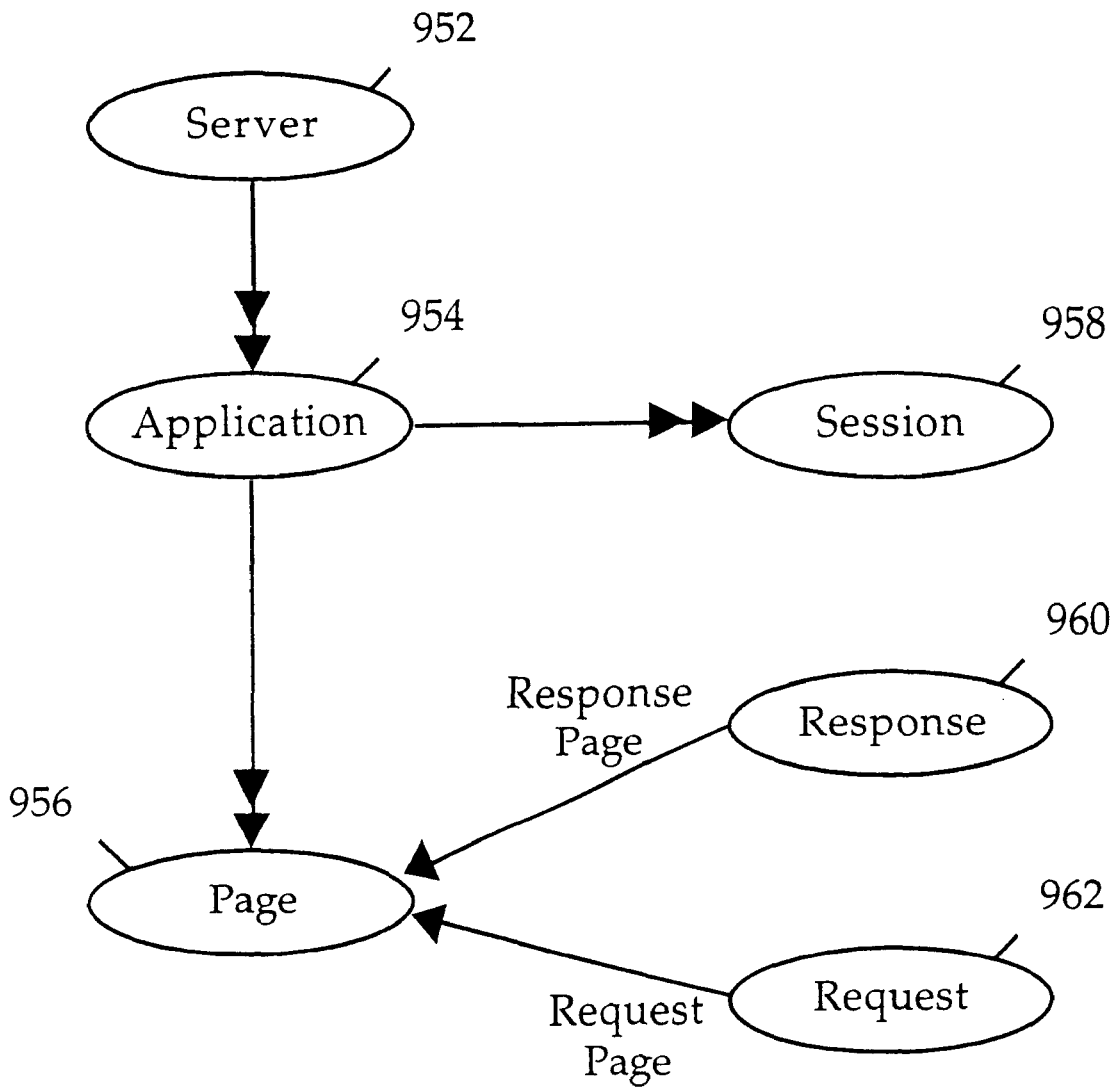

952
Server
954
Application
958
Session
960
Response
Page
Response
956
Page
962
Request
Request
Page

Figure 9B

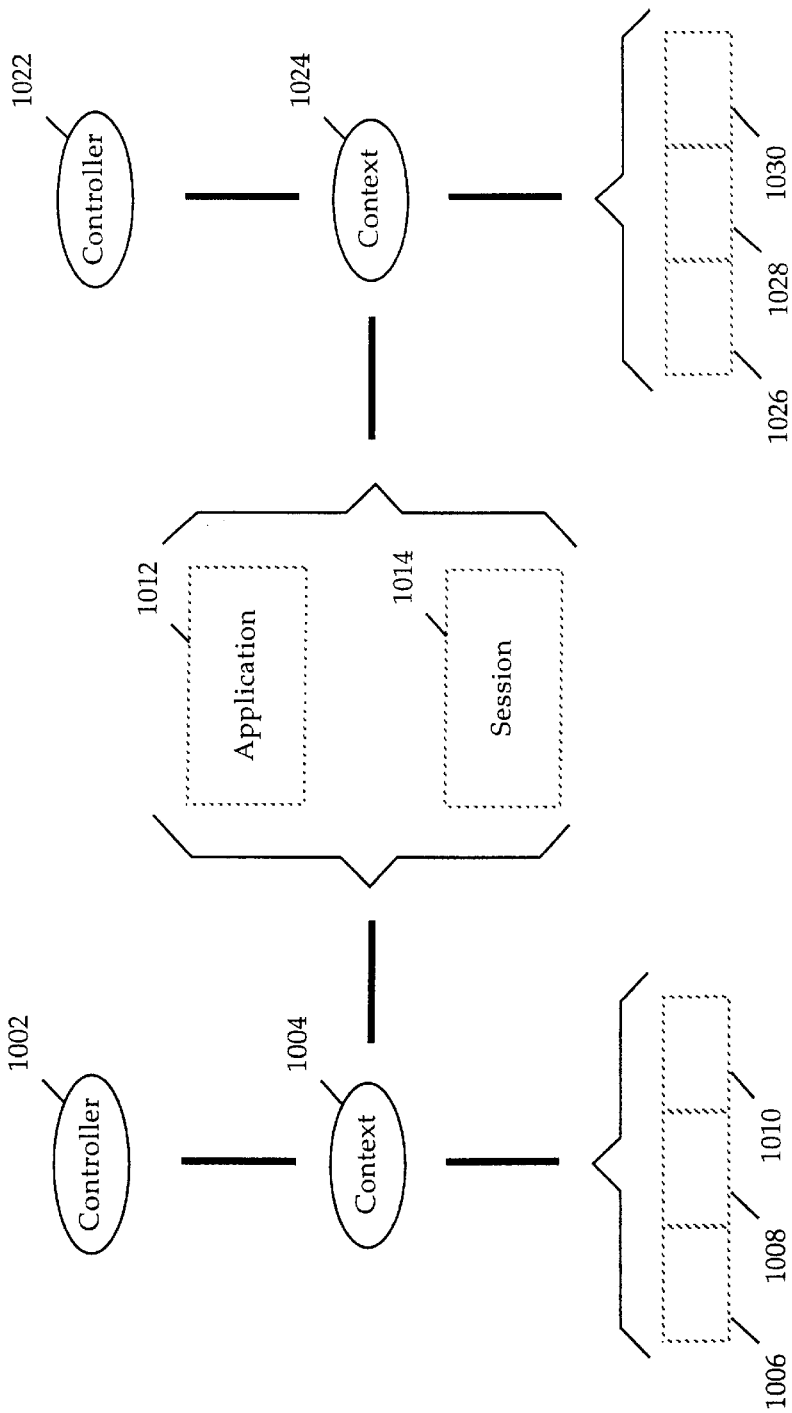

1022
Controller
1024
Context
1030
1028
1026
1012
Application
1014
Session
1002
Controller
1004
Context
1010
1008
1006

Figure 10

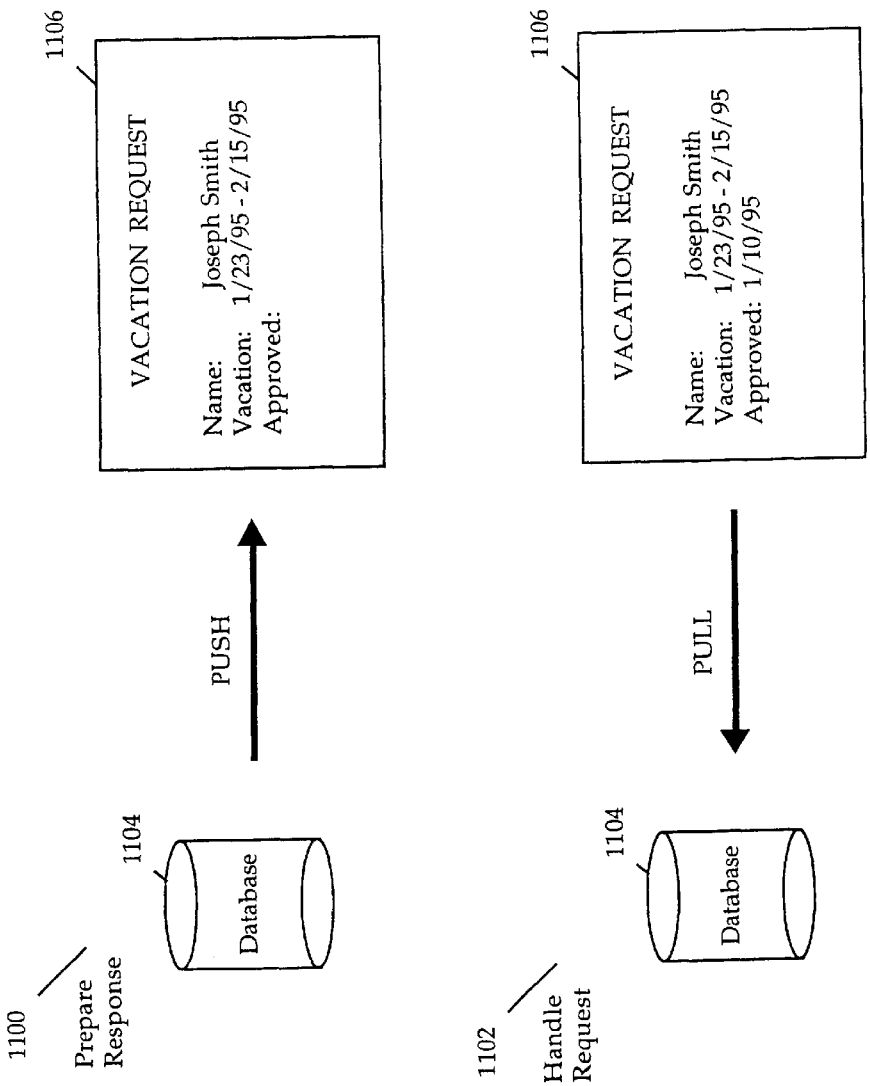

1106
VACATION REQUEST
Name: Joseph Smith
Vacation: 1/23/95 - 2/15/95
Approved:
PUSH
1104
Database
1100
Prepare Response
1106
VACATION REQUEST
Name: Joseph Smith
Vacation: 1/23/95 - 2/15/95
Approved: 1/10/95
PULL
1104
Database
1102
Handle Request

Figure 11A

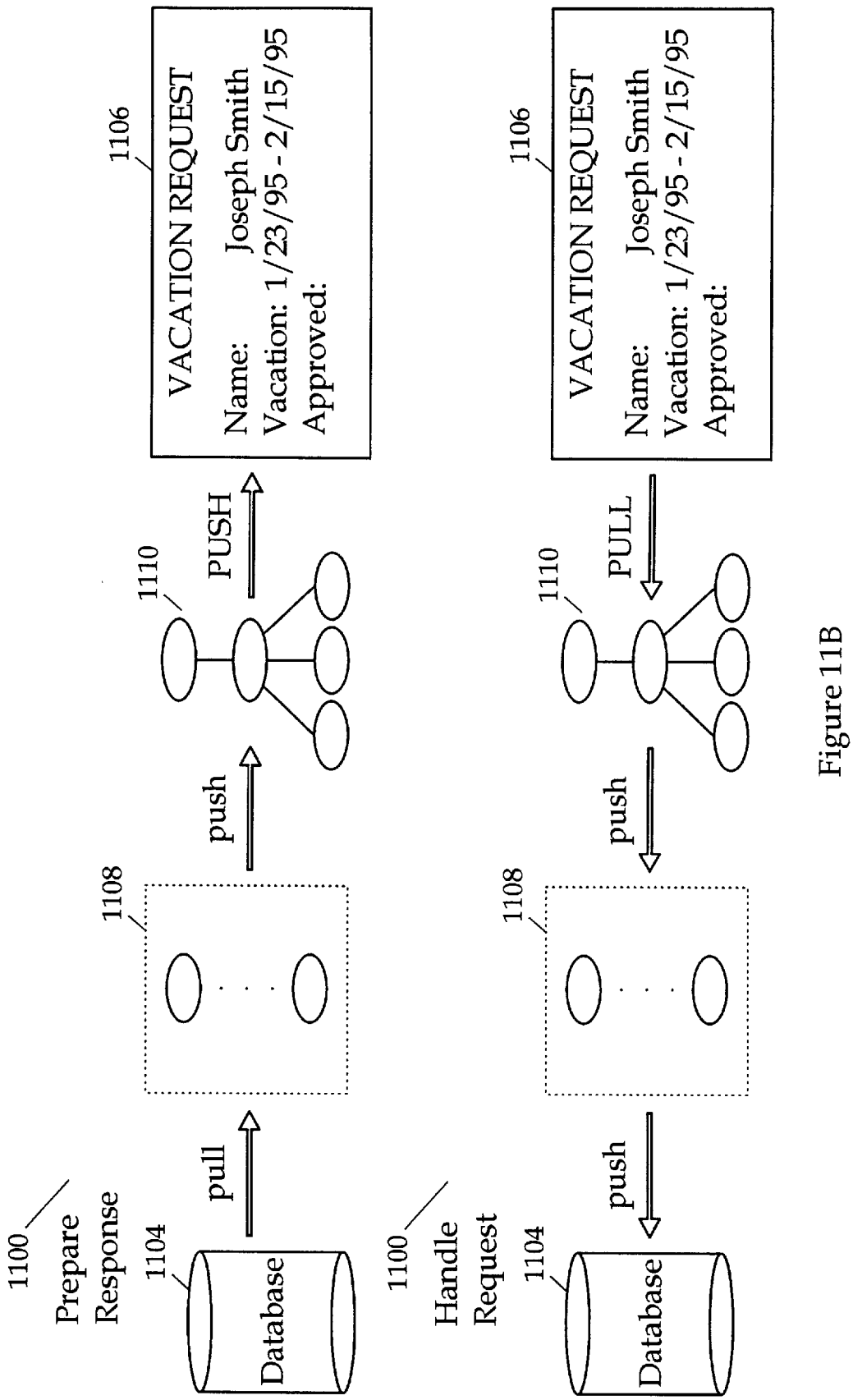

1100
Prepare Response
1104
Database
pull
1108
push
1110
PUSH
1106
VACATION REQUEST
Name: Joseph Smith
Vacation: 1/23/95 - 2/15/95
Approved:
1100
Handle Request
1104
Database
push
1108
push
1110
PULL
1106
VACATION REQUEST
Name: Joseph Smith
Vacation: 1/23/95 - 2/15/95
Approved:

Figure 11B

METHOD AND APPARATUS FOR MANAGING INTERNET TRANSACTIONS

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to the development and management of transactions on the Internet.

2. Background

The Internet is a worldwide matrix of interconnected computers. An Internet client accesses a computer on the network via an Internet provider. An Internet provider is an organization that provides a client (e.g., an individual or other organization) with access to the Internet (via analog telephone line or Integrated Services Digital Network line, for example). A client can, for example, download a file from or send an electronic mail message to another computer/client using the Internet.

To retrieve a file on the Internet, a client must search for the file, make a connection to the computer on which the file is stored, and download the file. Each of these steps may involve a separate application and access to multiple, dissimilar computer systems. The World Wide Web (WWW) was developed to provide a simpler, more uniform means for accessing information on the Internet.

The components of the WWW include browser software, network links, and servers. The browser software, or browser, is a graphical interface (i.e., front-end) that simplifies access to the Internet. A browser allows a client to communicate a request without having to learn a complicated command syntax. A browser typically provides a graphical user interface (GUI) for displaying information and receiving input. Examples of browsers currently available include Mosaic, Netscape, and Cello.

Information servers maintain the information on the WWW and are capable of processing a client request. Hypertext Transport Protocol (HTTP) is the standard protocol for communication with an information server on the WWW. HTTP has communication methods that allow clients to request data from a server and send information to the server.

To submit a request, the client contacts the HTTP server and transmits the request to the HTTP server. The request contains the communication method requested (e.g., GET an object from the server or POST data to an object on the server). The HTTP server responds to the client by sending a status of the request and the requested information. The connection is then terminated between the client and the HTTP server. A connection between a client and an HTTP server therefore has four stages: 1) initiation, 2) request transmittal, 3) response transmittal, and 4) termination. A connection can only process a single transaction or interaction between the client and the HTTP server.

The HTTP server does not retain any information about the request after the connection has been terminated. HTTP is, therefore, a stateless protocol. That is, a client can make several requests of an HTTP server, but each individual request is treated independent of any other request. The server has no recollection of any previous request.

An addressing scheme is employed to identify Internet resources (e.g., HTTP server, file or program). This addressing scheme is called Uniform Resource Locator (URL). A URL contains the protocol to use when accessing the server (e.g., HTTP), the Internet domain name of the site on which the server is running, the port number of the server, and the location of the resource in the file structure of the server.

The WWW uses a concept known as hypertext. Hypertext provides the ability to create links within a document to move directly to other information. To activate the link, it is only necessary to click on the hypertext link (e.g., a word or phrase). The hypertext link can be to information stored on a different site than the one that supplied the current information. A URL is associated with the link to identify the location of the additional information. When the link is activated, the client's browser uses the link to access the data at the site specified in the URL.

If the client request is for a file, the HTTP server locates the file and sends it to the client. An HTTP server also has the ability to delegate work to gateway programs. The Common Gateway Interface (CGI) specification defines the mechanisms by which HTTP servers communicate with gateway programs. A gateway program is referenced using a URL. The HTTP server activates the program specified in the URL and uses CGI mechanisms to pass program data sent by the client to the gateway program. Data is passed from the server to the gateway program via command-line arguments, standard input, or environment variables. The gateway program processes the data and returns its response to the server using CGI (via standard input, for example). The server forwards the data to the client using the HTTP.

A browser displays information to a client/user as pages or documents. A language is used to define the format for a page to be displayed in the WWW. The language is called Hypertext Markup Language (HTML). A WWW page is transmitted to a client as an HTML document. The browser executing at the client parses the document and produces a displays a page based on the information in the HTML document.

HTML is a structural language that is comprised of HTML elements that are nested within each other. An HTML document is a text file in which certain strings of characters, called tags, mark regions of the document and assign special meaning to them. These regions are called HTML elements. Each element has a name, or tag. An element can have attributes that specify properties of the element. Blocks or components include unordered list, text boxes, check boxes, radio buttons, for example. Each block has properties such as name, type, and value. The following provides an example of the structure of an HTML document:

```
<HTML>
  <HEAD>
  . . . element(s) valid in the document head
  </HEAD>
  <BODY>
  . . . element(s) valid in the document body
  </BODY>
</HTML>
```

Each HTML element is delimited by the pair of characters "<" and ">". The name of the HTML element is contained within the delimiting characters. The combination of the name and delimiting characters is referred to as a tag. Each element is identified by its tag. In most cases, each element has a start and ending tag. The ending tag is identified by the inclusion of an another character, "/" that follows the first delimiting character ("<").

HTML is a hierarchical language. With the exception of the HTML element, all other elements are contained within another element. The HTML element encompasses the entire document. It identifies the enclosed text as an HTML document. The HEAD element is contained within the HTML element and includes information about the HTML document. The BODY element is contained within the

HTML. The BODY element contains all of the text and other information to be displayed. Other elements are described by any HTML reference manual.

Web-based access has been used for simple information publishing that consisted of static data and no logic. For example, a client request might request another page. The HTTP server fetches a pre-defined Web page using the URL contained in the request. The Web page is returned to the client. Web-based access has progressed to dynamic information publishing that allows some simple queries and some limited logic to access a database and display database information. Web applications have been used in commerce to order articles of manufacture via the Web (e.g., a personal stereo cassette deck). Commerce applications provide some limited logic and allow a Web page to change based on the client input.

Web-based applications are developed independent of other Web-based applications. That is, constructs used to implement one Web-based application remain local to the application. They are not shared.

Currently, a common gateway interface program is written to process a single transaction. Minimal state information is retained within the HTML document. It would be beneficial to use object-oriented HTML elements to manage a series of Internet transactions and to maintain state information across Internet transactions. It would be beneficial to be able to generate HTML documents dynamically using information from a previous transaction, corporate data, and data retained in HTML element objects. It would be beneficial to be able to share application development across applications.

SUMMARY OF THE INVENTION

The object-oriented approach used by the present invention provides the ability to develop and manage Internet transactions. An HTML document can be generated dynamically. Some or all of a form can be generated based on input received in a receive form, generated at runtime, or retrieved from an external data source (e.g., database or electronic mail system). When the HTML document is rendered, the information contained in HTML template(s), declaration file(s), and object(s) are used to generate standard HTML.

Information regarding a client request is maintained by the application that processes the client request. The information is maintained in a state, or context. Different levels of state information are maintained by the present invention. The state level defines the duration and scope of the state information. Storage policies define the method of storage used for state information. By maintaining a context having levels, it is possible to form a virtual session with each user accessing an application such that an application can serve any request in any order.

Using the present invention, a user can connect to a local Internet provider and use the Web to connect to a home page. Once the connection is established, the present invention is used with an application on the server side of the connection to dynamically generate Web pages. The Web pages contain application information and provide the ability for the user to specify input. A browser executing on the user's workstation receives a Web page from the Internet and sends a response containing user input via the Internet. The present invention can be used to generate Web pages in response to the user input. Thus, a user can access an application located anywhere relative to the application server by using any local workstation hardware and software platform.

When a client request is received by the application, the application manages the update and retrieval of data from the context, processes the received form (i.e., the form received from the client that includes the client request), and processes any return form (i.e., a form sent to the client in response to the client request).

All or some portion of the Web page is dynamically generated. Classes of object are associated with elements that defined a Web page (e.g., HTML elements). By objectifying Web pages it is possible to gain the advantages of object-oriented programming with Web pages. These advantages include rapid application development through object reuse, transparent distribution, simplified maintenance, and scalability. These advantages can be achieved using the common interface provided by Web browsers.

Object instances associated with definitional elements are instantiated during execution to manage their respective portions of the Web page. Control objects are associated with definitional elements to facilitate the flow of information to and from external data sources.

Definitional elements can be grouped together. A group object class provides the ability to access these definitional elements as a group. A special type of control referred to as a repetition control allows a group to be activated multiple times. Each activation of a group of definitional elements generates one occurrence of a repeating portion of the Web page.

Self-contained modules, or components, can be accessed to provide additional definition for the Web page. A control object is associated with one of these modules to manage its integration into the rest of the Web page definition. The self-contained modules can be shared by one or more Web pages in a single application and/or across multiple applications executing on application server **316** (or any other server).

A page definition has some combination of the following associated resources: template(s), association(s), and custom logic. A template contains an initial definition for some or all of the page. At runtime, some or all of the initial definition can be dynamically altered. Associations provide declaration and binding information. Custom logic provides custom functionality.

A reusable or sharable construct of the present invention is referred to as a component. A component can represent an entire page or some portion of a page. A component has none or more resources. A resource can be, for example, a template, association, or custom logic. A component owns its associated resources.

One or more components can be combined to define a page. A component can be defined using different definitional constructs (e.g., HTML, VRML, PDF, etc.). By combining these components as a single Web page, a document can be derived from multiple definitional constructs.

A control class provides object cover for components. An abstract class provides the ability to: associate a Web object (e.g., an HTML element object) with a corporate object, generate static elements, and modify an object tree. Subclasses of the control class inherit behavior from the abstract class. Instances of these subclasses provide push and pull mode associative behavior. In push mode, data retrieved from an external data source (or another source) is used to dynamically generate a Web page. In pull mode, data contained in the Web page can be stored into the context or corporate object.

A control object can have subcontrols that are themselves control objects. Subcontrols can be pre-defined or generated at runtime. The associative behavior of a subcontrol is attributed to the parent control object. A control object

activates a subcontrol's push and pull modes by forwarding the activation message that it received.

Event objects are used to handle a client request. These event objects contain request information and behavior to verify and process an client request. Custom logic can be invoked during event handling. The application manages the process and event objects.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** provides an example of a general purpose computer to be used with the present invention.

FIG. **2** provides a block-level diagram that illustrates components of the present invention.

FIG. **3**A provides a block-level diagram for accession a corporate application on the World Wide Web (WWW) using the present invention.

FIG. **3**B provides an example of Web pages generated by an Automobile Shopper's application accessed via the WWW using the present invention.

FIG. **4** provides a diagram of the object tree.

FIGS. **5**A–**5**E provides an example of a process flow for handling a client request.

FIG. **6**A provides an example of resources associated with a control.

FIGS. **6**B–**6**C provide a block-level diagram associated with component processing.

FIG. **7**A provides the object tree that is generated when the HTML in Table Four is parsed.

FIG. **7**B provides an example of the object tree of FIG. **7**A and an associated repetition controller and repetition template.

FIG. **7**C provides an example of the object tree of FIG. **7**B that has been modified using a repetition controller and template.

FIG. **8** illustrates a rendering process flow.

FIGS. **9**A–**9**B illustrate event handling object diagrams.

FIG. **10** illustrates context objects.

FIGS. **11**A–**11**B illustrate the push and pull modes of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A method and apparatus for developing and managing Internet transactions is described. In the following description, numerous specific details are set forth in order to provide a more thorough description of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well-known features have not been described in detail so as not to obscure the invention.

The present invention can be implemented on a general purpose computer such as illustrated in FIG. **1**. A keyboard **110** and mouse **111** are coupled to a bi-directional system bus **118**. The keyboard and mouse are for introducing user input to the computer system and communicating that user input to CPU **113**. The computer system of FIG. **1** also includes a video memory **114**, main memory **115** and mass storage **112**, all coupled to bidirectional system bus **118** along with keyboard **110**, mouse **111** and CPU **113**. The mass storage **112** may include both fixed and removable media, such as magnetic, optical or magnetic optical storage systems or any other available mass storage technology. Bus **118** may contain, for example, 32 address lines for addressing video memory **114** or main memory **115**. The system bus **118** also includes, for example, a 32-bit DATA bus for transferring DATA between and among the components, such as CPU **113**, main memory **115**, video memory **114** and mass storage **112**. Alternatively, multiplex DATA/address lines may be used instead of separate DATA and address lines.

In the preferred embodiment of this invention, the CPU **113** is a 32-bit microprocessor manufactured by Motorola, such as the 680X0 processor or a microprocessor manufactured by Intel, such as the 80X86, or Pentium processor. However, any other suitable microprocessor or microcomputer may be utilized. Main memory **115** is comprised of dynamic random access memory (DRAM). Video memory **114** is a dual-ported video random access memory. One port of the video memory **114** is coupled to video amplifier **116**. The video amplifier **116** is used to drive the cathode ray tube (CRT) raster monitor **117**. Video amplifier **116** is well known in the art and may be implemented by any suitable means. This circuitry converts pixel DATA stored in video memory **114** to a raster signal suitable for use by monitor **117**. Monitor **117** is a type of monitor suitable for displaying graphic images.

The computer system described above is for purposes of example only. The present invention may be implemented in any type of computer system or programming or processing environment. When a general purpose computer system such as the one described executes the processes and process flows described herein, it is a means for developing and managing Internet transactions.

The present invention provides the ability to access an application using any interface. For example, a client can access the same application via the Internet using a system running Windows, MAC/OS, Sun.OS, NextStep, etc. Referring to FIG. **2**, client **202** is resident on a corporate network **200**. Corporate network **200** is a local area network comprised of personal computers such as client **202**, for example. Client **202** executes a browser such as Netscape or Mosaic to interface with World Wide Web (WWW) **204**. The browser executing on client **202** displays HTML documents transmitted to client **202** via WWW **204**. When the client browser submits a request typically an HTML page, it is transmitted via WWW **204** to HTTP server **206**. An attribute of the submitted form is a Universal Resource Locator (URL). The URL specifies procedures to be executed and/or data input by the user.

HTTP server **206** can service the client request by, for example, retrieving a return form or file (specified by a URL contained in the HTML document) and transmitting it to client **202** via WWW **204**. Further, HTTP server **206** can initiate a program identified in the URL. In the preferred embodiment, the URL specifies CGIMessenger **210** to service the client request. CGIMessenger **210** uses the Common Gateway Interface (CGI) protocol, for example. CGIMessenger **210** can execute on the HTTP server **206** or another server connected to HTTP server **206**, for example. The following provides an example of a URL:

```
http://www.next.com/cgi_bin/cgimessenger/app_name/rcv_form/
send_form
```

The above example is for illustration only. Any URL format can be used with the present invention. The URL in the above example has multiple parts. The first part, "http:" specifies the protocol to be used. Other examples of protocol include "ftp:", "gopher:" and "wais". The next part specifies

the Internet domain name of the server. In this case, the Internet domain name of the HTTP server is "www.next-.com". The next part, "/cgi_bin/cgi_messenger" specifies the path and filename of the desired resource. The resource can be a file, form, or program, for example. In this case, the resource is a common gateway interface program that is used to route requests to applications. The application that processes the client request is specified in the URL with the "app_name" specification. The names of the receiving form (e.g., the form containing the client request) and the sending form (e.g., a reply to the client request) can be included in the URL.

A client request can result in the invocation of application **214** in the server domain **208**, for example. The client request is received by the HTTP server **206**. HTTP server **206** initiates CGIMessenger **210**. To service the client request, CGIMessenger **210** communicates with HTTP server **206** to obtain information regarding the client request. For example, CGIMessenger **210** obtains any information that accompanied the user request such as form and/or application names and/or user input. The form names can include the name of the form that was submitted to initiate the client request as well as a return form.

The application name identifies the application that services the client request. If an application is specified, CGIMessenger **210** transmits the client request and corresponding information transmitted from HTTP Server **206** to application **214**. Application **214** can execute on the same or different server as CGIMessenger **210** and/or HTTP Server **206**, for example. Application **214** executes an interaction flow to satisfy the user request. Application **214** can access an external data source such as database **224**. Database **224** can be resident on the same server as application **214**. Alternatively, database **224** can be resident on a separate server (e.g., a separate database server).

The present invention can service multiple levels of interaction with the client browser. A simple hypertext request (e.g., a request for another form specified in the request) is serviced by HTTP Server **206** by returning the specified form. A simple hypertext request can be, for example, a request for a static HTML page (e.g., a page that contains static information that is defined prior to execution) that does not have any associated logic.

The present invention provides the ability to satisfy more sophisticated requests. For example, the present invention can be used to access a Web page (e.g., an HTML Web page) that is dynamically generated using complex queries (or other data retrieval mechanisms) to retrieve data and dynamically generate an HTML page using complex logic. For more sophisticated requests, the present invention can execute logic such as CGIMessenger **210** and application **214**, for example, to process a client request. The application preferably includes an object class hierarchy that can be implemented in any object-oriented language such as objective C, SmallTalk or JAVA. Objects (e.g., Web page definitional objects and control objects) dynamically generate a Web page and manage client interaction via the Internet and WWW.

Internal Application

A powerful feature of the WWW is its accessibility virtually anywhere in the world using virtually any hardware/software platform. Using the present invention, this feature of the WWW can be used to allow an application such as a corporate (or local) application to be accessed using any workstation that provides access to the Web regardless of its location. The present invention can be used to allow an internal, corporate application to be accessed any place in the world via any hardware/software platform that is capable of accessing the Internet.

The present invention provides tools to implement an application that can be accessed via the WWW. Further, the present invention provides tools to manage the interaction between a WWW client and the internal application. FIG. **3**A provides a block overview of an internal application that is accessed via the Internet.

Clients **302**, **304**, **306**, and **308** access WWW **310** using any WWW-capable workstation. For example, client **302** can be an Intel-based workstation running Windows and a browser such as NetScape Navigator. Client **304** can be a 68K workstation running MacOS and Mosaic WWW browser. Client **306** can be an RISC machine running UNIX and a proprietary browser. Client **308** is an Intel-based workstation running DOS and the DosLynx browser.

Requests submitted by clients **302**, **304**, **306**, and **308** to HTTP server **314** are transmitted to HTTP server **314** via WWW **310**. HTTP server **314** runs in the corporate network **312**. A client request is forwarded to application server **316** via HTTP server **314**. Application server **316** runs one or more internal applications.

The present invention accommodates the needs of an internal application. For example, using the present invention an internal application can provide a dynamic user interface that can respond to user input. Further, an internal application is able to access an external data source to store the application's data. State information can be retained to allow the internal application to allow the application to maintain open sessions with multiple users capable of processing multiple transactions in any order.

Development of an internal application can be efficient and flexible. An internal application can be written using complex logic and complex queries. Developers have the ability to incorporate pre-existing modules into the internal application. Modules are self-contained to facilitate module maintenance and interaction.

An internal application running on application server **316** has the ability to respond to the client request using the capabilities of the present invention. The Web page definition returned to a client can be generated from a pre-defined template or templates. Alternatively, all or some portion of the definition can be dynamically generated by the application. The contents of the Web page can contain data retrieved from an external data source (e.g., such as database server **318**), or state information maintained by the application. Custom logic and declarations can be associated with each Web page.

The application can make use of object technology to generate client responses. For example, the elements of a Web page definition are objectified. Instances of definitional elements are instantiated during execution to manage their respective portions of the Web page. Control objects are associated with definitional elements to facilitate the flow of information to and from external data sources. A special type of control referred to as a repetition control allows a group to be activated multiple times. Each activation of a group of definitional elements generates one occurrence of a repeating portion of the Web page.

Definitional elements can be grouped together. A group object class provides the ability to access these definitional elements as a group.

Self-contained modules, or components, can be accessed to provide additional definition for the Web page. A control object is associated with one of these modules to manage its integration into the rest of the Web page definition. The self-contained modules can be shared by one or more Web

pages in a single application and/or across multiple applications executing on application server **316** (or any other server).

Internal application **320** can be, for example an Automobile Shopper's application. It can be used by a prospective car buyer to select a car. A car shopper accesses the Automobile Shopper's application. The application provides a series of screens (i.e., Web pages) based on user-input that are designed to facilitate the selection and purchase of an automobile. FIG. **3**B provides Web pages that are generated using application **320** and displayed at a client workstation such as clients **302**–**308**.

The content of the Web pages seen by the shopper are determined based on the shopper's input. A Web page can be generated from a pre-built or static definition. However, it would be necessary to store a definition to accommodate each selection or combination of selections made by the shopper. The use of static page definitions may be acceptable where the number of pages is limited. However, where the input can result in a wide variety of Web pages, the use of static definitions is very inefficient. For example, each static definition must be stored and catalogued for later access. Where the content of the page can vary based on the input, the use of dynamic page definitions is more efficient. There is no need to store a definition for every possible Web page when a page is dynamically generated.

Referring to FIG. **3**B, the Automobile Shopper application provides an example of the use of dynamic pages. "Welcome" page **332** provides input areas that allow the shopper to identify model, price and type of the car(s) in which the shopper has some interest. After specifying the characteristics, the shopper submits the page. In response, the shopper expects a page that displays those cars that meet the criteria specified by the shopper in "Welcome" page **332**. The response page is based on the particular input provided by the shopper.

HTTP Server **314** receives the "Welcome" page. The "Welcome" page contains a URL that specifies Automobile Shoppers application **320**. HTTP Server **314** forks a CGI interface program **322** (e.g., CGIMessenger). CGI program **322** identifies the application specified in the URL. CGI program **322** passes the shopper's input contained in the "Welcome" page to the application **320** running on application server **316** via corporate network **312**.

Application **320** receives the user-specified characteristics. Custom logic associated with Welcome page **332** can be used to determine the next page to be sent to the shopper.

Objects associated with the next page are initiated. Custom logic associated with the next page is executed to build a select statement to retrieve data from an external data source (e.g., database server **318**). To fetch the data from the external data source, the application builds a qualifier object, for example. A data source object acts as an interface to the database. A data source object has fetch, insert, update, and delete operational capabilities.

The application sends the qualifier object and fetches all the data from the database. An array of data is returned from the fetch operation. For example, the array contains all of the models of cars that are within the price range and type specifications provided by the shopper in the "Welcome" page. Using the data retrieved from database server **318**, the application along with the objects associated with the next Web page generate the definition for Web page **334**.

Web page **334** contains a repeating portion that is comprised of an image, model data, type information, pricing. A group can be used to generate the repeating portions of Web page **334**. The group generates a template that can be used to generate each repetition. The group includes objects that correspond to definitional elements in the repetition. The group is activated for each repetition. When the group is activated, the objects contained in the group are activated. The objects generate the definitional elements (e.g., HTML statements) that define one repetition. Referring to page **334**, an activation activates a group and the objects contained in the group to generate HTML statements that include definitions for the image, string, and input fields contained in the first half of the page. A second activation of the group generates the bottom portion of Web page **334**.

The shopper selects one of the models by clicking on the image of the car. The shopper expects more detailed information about the specific model selected in response to the selection. In response to the shopper's selection, the application and objects associated with the Web page are used to fetch information about the selection from database server **318**. The objects generate the definitional elements that are used by the shopper's browser to display Web page **335**.

A shopper selects an option package (e.g., "Package D" or "Package F"), select a color for the car, or ask for a screen that contains a list of dealers that have the selected car in stock. For example, if the shopper selects a color for the car, the custom logic associated with page **335** determines that the same page definition can be used to generate a response page. An image of the car in the color selected by the shopper can be inserted into the page definition and sent to the shopper's workstation. The browser executing on the shopper's workstation displays Web page **336**.

The Web pages in FIG. **3**B are generated dynamically based on a shopper's input. A Web page in FIG. **3**B can be generated using resources associated with the page. For example Web page **332** can have an associated set of resources. These resources can include none or any combination of: template(s), association(s) and custom logic.

A template provides an initial definition for the Web page. It is used to generate a tree of objects that can be used to generate the Web page. The template and corresponding object tree can be modified dynamically based on the associations and custom logic. An association can provide a declaration of a definitional element that is to be inserted into the object tree. An association can further be used to modify properties of objects in the object tree. The custom logic is preferably implemented using a script or interpreted language.

Script

The application (e.g., application **214** in FIG. **2** or application **320** in FIG. **3**) can make use of procedures to service a client request. In the preferred embodiment, application **214** uses objects and other procedures to provide logic to satisfy a client request. A scripting language can provide logic, for example. In addition, object instances can provide behavior to process a client request.

A script is a type of program that consists of a set of instructions that is read and interpreted by another application. An instruction is an expression that conforms to the rules and syntax expected by a script interpreter. A script language (set of script instructions) may include simple control statements such as loops and if/then expressions.

Since the language does not include complex constructs, the language is easy to master. As with other interpretive languages, a developer does not have to perform steps to translate a script into machine-readable code. Even though a script is interpreted at runtime, it is not considered to cause any degradation in the overall response time (the actual network response times are the primary concern). Therefore, scripts are typically used to provide the logic associated with a Web page.

Practical Extraction and Reporting Language (PERL) is one example of a scripting language. Alternatively, an objective C (interpreted or compiled) can be used. An interpretive objective C is used in the preferred embodiment. However, any programming language can be used with the present invention.

Element Objects

Objects provide behavior and data to facilitate a client request. Preferably, object classes are defined to objectify the elements that define a Web page. For example, a Web page that is defined using HTML can be generated using an object class library that includes HTML element object classes. The present invention is described herein with reference to HTML, however, object classes can be defined to accommodate any type of definitional constructs or syntax that can be used to define a Web page (e.g., JAVA, PDF, VRML, etc.).

By objectifying Web pages it is possible to gain the advantages of object-oriented programming with Web pages. These advantages include rapid application development through object reuse, transparent distribution, simplified maintenance, and scalability. These advantages can be achieved using the common interface provided by Web browsers.

Objectifying HTML elements provides the ability to programmatically manipulate HTML documents. The properties of each element are stored in instance variables of the associated object. Each object class can include methods to manipulate the HTML element within an HTML document. The mapping of HTML elements to objects is more fully described in a co-pending U.S. Patent Application entitled "Method and Apparatus for Generating Object-Oriented World Wide Web Pages", Ser. No. 08/515,057, filed on Aug. 15, 1995, assigned to the assignee of the present invention, and incorporated herein by reference.

The HTML elements that define a Web page can be included in a single ASCII file. A file that contains HTML elements is referred to as an HTML template. An HTML template can include a complete definition of the Web page. However, an HTML template may define a subset of the Web page. Two or more HTML templates that define portions of the Web page can be combined to provide a complete definition. The following provides an example of the contents of an HTML template:

TABLE One

```
<HTML>
    <BODY>
    This is a list of employees:
    <P>
    <OL>
    <LI> James </LI>
    <LI> Mary </LI>
    </OL>
    </BODY>
</HTML>
```

The HTML template in Table One provides one example of a template that can be used with the present invention. It includes the HTML elements: HTML, BODY, Paragraph (P), Ordered List (OL), and List Item (LI). The present invention defines a class for each HTML element. An object class defines instance variables to store information associated with the HTML element such as attributes, or properties, of an HTML element. Methods can be defined to manipulate an HTML element to render (i.e., generate) an HTML document, for example.

Preferably, the objects in an object tree are instantiated based on pre-defined object classes. Object classes are defined to manipulate and generate HTML elements in an HTML document. The name of the object class can mirror the names of the corresponding HTML element to provide an easy association between the two. Alternatively, any mapping scheme can be used to map an HTML element to an object class. Instance variables can be used to store property information defined by properties in the HTML element statement. An object class can further contain instance variable declarations to store other information associated with an HTML element.

An object class can have behavior that can be used to manipulate the HTML document. For example, an object class may have a method that varies the display characteristics of the HTML element based on the outcome of some Boolean logic. Further, instance variables can be used to provide a pointer to scripts used with the HTML page.

Referring to FIG. **2**, application **214** includes objects **216**. Objects **216** correspond to the HTML elements that define a WWW page. Objects **216** are arranged in a tree structure that corresponds to the hierarchical structure of the HTML elements that they implement. To render an HTML document, application **214** sends a message to the root object **218** to render itself. In response to this message, root object **218** renders the corresponding HTML element(s). Root object **218** sends a message to its children, objects **220**A–**220**B, to render themselves. This process is continued until all of the nodes in the object tree have been processed.

Appendix A provides a list of some object classes for HTML elements. Additional or modified class definitions can also be used with the present invention. A library of object classes are preferably pre-defined. The object classes described above are used in conjunction with HTML elements. Other object classes can be used with the present invention to accommodate other definitional elements used in Web page that includes JAVA and PDF, for example.

State

A web session has consisted of a series of independent transactions. It is stateless. HTML provides a mechanism to store a limited amount of state information as part of the HTML page. The present invention provides a mechanism to store state information independent of the Web page. State information maintained by the present invention facilitates the use of the WWW as a vehicle for accessing a local application, for example. The states provided by the present invention provide the ability to implement virtual applications, sessions, transactions, etc.

State, or context, typically refers to the condition of processing and/or information at a given instance. In the preferred embodiment of the present invention, state information has associated duration and scoping designations. State information is preferably contained in the properties of objects. Objects are said to be contained within the state. A state's persistence relates to the persistence of its objects. Object persistence relates to the degree of permanence of an object. For example, an object can be created for the duration of an application's execution, or for some shorter period of time. Table Two provides examples of persistence levels and the corresponding context duration and scope that are preferably used with the present invention:

TABLE Two

Level	Duration	Scope
Local	The life of an object is the same as that of the procedure or method in which it was created	Accessible within the procedure or method
Transaction	An object exists during the processing to satisfy a client request (e.g., generate and transmit a Web page)	Accessible by objects involved in the current transaction(s)
Page	An object exists for the duration of transaction (e.g., an object exists during the generation of a page and is available to process the page when it is returned by the client)	Accessible by objects involved in the current transaction(s) or in the next transaction(s) of the same session
Session	Once an object is created in a client session, it remains for the remainder of the session	Accessible by objects involved in any transacfion of a session
Global	An object created by an application exists for the remainder of the application's processing	Accessible by objects involved in any transaction of any session of an application

The context levels provided in Table Two are for purposes of illustration. Additional or different levels supplement or replace these levels to define state and/or object duration and scope without departing from the scope of the invention.

In the preferred embodiment, state information is retained in properties of objects. By specifying the level of persistence of an object containing state information, the level of persistence of the state information is also specified. State information is maintained at the local, transaction, page, session, or global levels, for example. A client's login password can be retained as state information at the session level, for example. Thereafter, the client does not have to enter the password to access an application during that session. Global level information is shared across sessions, clients, and applications. For example, a connection to a database can be used at the global level by multiple sessions and applications. One of the levels (e.g., local) can be used as a default that is used when a level is not specified.

Each level has an associated scope that defines the scope or accessibility of the level's context information. For example, global-level context information is accessible by objects involved in any transaction within any session of the application. Session-level context information is accessible by objects that are involved in the processing of a transaction within the particular session. Objects that process a current or next transaction (i.e., the transaction that follows the current transaction) have access to page-level context information associated with the current transaction's page. Objects involved in the generation of a Web page can access that transaction's transaction-level information. Context information that is local to a procedure or method is accessible within that procedure or method.

By maintaining context levels, it is possible to maintain context information that is unique for each application and session within each application. Thus, it is possible, for example, to form virtual applications and sessions, for example. For example, context information associated with a session can be designated as session-level context information thereby making the information accessible to objects or other logic that are involved in any transaction performed within the session. Each session can therefore have its own context information. An application can retrieve a particular session's context information to process a request. Similarly, the same application can access another session's context to process another request.

Various techniques can be used to specify a context level. In the preferred embodiment, a context level is specified in the declaration section of a scripted procedure or other logic. The particular syntax used in the declaration is dependent on the language used to define the logic. The following provides an example of declarations using an objective C syntax:

```
transaction a;
page b, c;
global t, u, v;
```

In the above examples, the keywords "transaction", "page", and "global" define the level of persistence for the variables that follow the keywords. Thus, for example, the object "a" has a "transaction" level of persistence. Therefore, "a" exists during the processing of a client request, for example. Objects "b" and "c" have "page" level persistence and remain in existence to receive a page returned from a client, for example. The result of the declaration statement is to put objects "b" and "c" in the page-level context. The "global" objects "t", "u" and "v" are accessible for the life of an application and are shared among sessions, for example.

In the examples above, the level of persistence is specified in a declaration statement. Alternatively, the level of persistence can be defined using APIs in interpretive C, for example. A context object is used to manage different pools of objects (i.e., global pool, session pool, transaction pool, etc.).

A "contextSetObjectForKey" message is sent to the context object. The message tells the context object to set an object (named in the message) in an object pool according to a context level specification. If a context level is not specified, the context object searches the object pools for an object with the same name. If it does not find an object with the same name and no context level is specified, the context object inserts the object in the default pool (e.g., local context level).

The context retrieves objects from the object pools using criteria such as object name and pool designation. The context retrieves the object(s) for the requester. The context acts as the manager of the name space.

This functionality is also implemented by the script interpreter used in the preferred embodiment when a context-level keyword (e.g., "global, page, transaction, etc.) is encountered.

State information is retained using a storage means. Various storage policies can be used with the present invention. The actual means of storage is preferably programmatically alterable. A storage policy can be specified programmatically on a context basis or on an object basis, for example. A default storage policy can be set for all of the objects that have session-level persistence, for example. The default policy can be overridden by a policy specified for an object having session-level persistence. Table Three contains examples of storage policies:

TABLE Three

Policy	Description
Page	State information is archived with the Web page using an HTML "hidden" field
Database	State information is archived in an external data source such as a database

TABLE Three-continued

Policy	Description
Memory	State information is archived in memory
File System	State information is archived to a file system

Other storage policies in addition to, or as a replacement for, those specified in Table Three can be used with the present invention. Since the state is preferably stored in objects, a storage policy specifies the archival policy used for an object containing state information. A "page" storage policy results in the object using this policy to be archived within a Web page in HTML "hidden" fields, for example. Alternatively, a token can be stored in the Web page that can point to the object(s), for example.

Objects are archived in a database using a "database" storage policy. A second storage policy can be used to store a database key to access the objects archived to the database. Using the "memory" storage policy, a state is associated with a unique client so that an application can service multiple sessions for multiple clients, for example. Using the "file system" storage policy objects are archived to disk storage, for example. Objects are stored in the file system by, for example, maintaining the objects and their corresponding properties in files in the file system.

Using any storage policy, it may be necessary to perform a garbage collection process to eliminate state information that is no longer needed. A time stamp is used to facilitate garbage collection. One method of garbage collection is described in U.S. Pat. No. 5,355,483 entitled Asynchronous Garbage Collection issued Oct. 11, 1994, assigned to the assignee of the present invention, and incorporated herein by reference.

Group Object

There is a standard set of HTML elements that are understood by most Web browsers. Some Web browsers offer additional HTML elements or extensions. These extensions provide additional functionality. However, because extensions are not part of the standard HTML, one browser's HTML extension may not be interpretable by another browser.

The extensions offered by client browsers are interpreted on the client side. The present invention provides an extension to HTML that is used on the HTTP server side. The HTML extension is filtered out before a Web page is sent to a client browser. It is used to interpret an HTML template and to render an HTML document before is transmitted to the client browser. Because HTML extensions are not sent to the client browser, there is no need for a client browser to support these extensions. Thus, existing client browsers that support the standard HTML can be used to display HTML documents using the present invention.

The present invention uses an extension to the standard HTML known as the group extension. The group extension provides the ability to combine HTML elements or statements in a single block. Further, the group extension provides a link between an object that implements an HTML element and an object that implements a data item stored in a data source external to the WWW application. A name attribute of the group extension provides the ability to identify the block, or group, by name. The name attribute can be stored in a hash table. During processing, the group identity can be retrieved and used to traverse an object tree.

The group extension also ensures the scoping of named elements (i.e., elements that have a name by virtue of a name attribute). A group object maintains a hash table that includes the named elements which are its members. The group element scopes the named elements within itself. Therefore, two elements having the same name in different groups are distinguishable. Each one is scoped to its respective group. For example, a group element named "Nico" contained within a parent group named "Employee" is distinct from a group named "Nico" within a parent group named "Tourist".

An NSWTAG marker is used to identify the bounds of the group in HTML. The following is an example of HTML including the group extension:

TABLE Four

```
<HTML>
    <H1 ><NSWTAG NAME="GREETING"></NSWTAG></H1>
    <NSWTAG NAME="SELECT_FORM">
        <FORM>
                    . . .
            <OL>
            <NSWTAG NAME="WEBPEOPLE">
                <NSWTAG NAME="FN"><INPUT TYPE=
                "text"></NSWTAG>
                <NSWTAG NAME="LN"><INPUT TYPE =
                "text"></NSWTAG>
            </NSWTAG>
            </OL>
                    . . .
        </FORM>
    </NSWTAG
                    . . .
</HTML>
```

The group extension is identified by the <NSWTAG> and </NSWTAG> delimiters. Other techniques can be used with the present invention to identify a group element in HTML, or other Web page definitional syntax. A NAME property specifies the name of the group. For example, a "GREETING" group extension is defined within the H1 HTML element. A group extension can contain other HTML definitions including other group definitions. For example, the "SELECT_FORM" group contains: FORM, OL (Ordered List), "WEBPEOPLE" group, "FN" group, INPUT, "LN" group, and INPUT HTML elements. The "WEBPEOPLE" group is nested within an OL HTML element within the "SELECT_FORM" group extension. The "LN" and "FN" group extensions are nested within the "WEBPEOPLE" group.

The name property of a group HTML element is used to link a group to an association.

Associations

An association is used in combination with the group extension. An association provides declaration information for static HTML elements where appropriate. It further provides binding information. An association is declared in an associations file.

An association can contain a declaration for an HTML object. Properties of the HTML object can also be defined within in the association. Values of properties declared in the association are used to populate the properties in an instance of the HTML object class.

An association provides binding information to bind, for example, variables, objects, Web page definitions, and scripts to one another. For example, binding information can bind an HTML object to an object that contains corporate data, or data retrieved from an external source (e.g., an Enterprise Object, or "EO"). An association binds a script with a Web page definition, for example. Binding information is preferably contained in an association as a name or label.

Table Five provides examples of associations for the group extensions defined in Table Four.

TABLE Five

```
SELECT_FORM:NSWForm {action=OnSelection};
. . .
SELECT_FORM.WEBPEOPLE:NSWRepetition {list=web_dept; item=employee};
SELECT_FORM.WEBPEOPLE.FN:NSWInput {value=employee.fn};
SELECT_FORM.WEBPEOPLE.LN:NSWInput {value=employee.ln};
. . .
GREETING:NSWString {value="Welcome"};
```

The association syntax provided in Table Five is for illustration purposes only. The particular syntax used can vary without departing from the scope of the present invention.

An association provides information that can be used with its corresponding group HTML element. The information contained in an association is preferably contained as fields. A label, or name, field (the string to the left of the colon in the above example) provides a link to the group declaration in the HTML template. An HTML object declaration field (the field to the right of the label field) identifies an HTML object class to instantiate. The HTML object class instance is inserted in the object tree based on the information contained in the label field. Property fields of the association contain values to populate properties of the HTML object instance.

When an HTML template is parsed, the HTML template and associations information along with the logic (e.g., in the application and scripts) are used to generate an object tree. FIG. **7**A provides an object tree that is generated when the HTML template in Table Four is parsed.

Page object **702** is the root of the object tree. A header object **704** (i.e., H1) is a child of page object **702**. The "GREETING" group object **708** is a child of header object **704**. The name attribute of the corresponding group HTML element provides a link to an association. Within page object **702**, is a group object **706** with a "SELECT_FORM" name attribute. The "SELECT_FORM" group object **706** contains FORM HTML object **710** and OL HTML object **714**. Within the OL HTML object **714** is a group object **716** with a name attribute equal to "WEBPEOPLE". The "WEBPEOPLE" group object **716** is instantiated from the group definition.

An association can be used to modify an object tree by adding objects to the tree or modifying the objects that already exist in the tree. For example, an HTML object, or element, declared in an association can be inserted in the object tree. Property values that are declared in the association can be used to modify an existing object's properties.

Component

A component is a construct that can be reused or shared (e.g., across applications). A component can have associated resources (e.g., template, associations, and custom logic) that are owned by the component. These resources are stored, for example, in a file system with a known search path. Alternatively, a component's definitions and logic can be determined dynamically at runtime. A web page can be defined using the resources owned by components. Components provide the ability to use multi-content document (i.e., documents generated using different types of definitional constructs).

A component is an independent element that owns its resources. It can have custom logic. A component can provide a definition for a Web page using a language such as HTML. Other languages or constructs used to define a Web page can also be used. Thus, for example a component can be defined using JAVA, PDF, VRML, or ACROBAT. Components can be combined to define a single Web page regardless of the language used to generate the definitions.

For example, a Web page can be defined using an HTML component (i.e., a component that contains HTML element definitions), a PDF component (i.e., a component that contains PDF), and a VRML component (i.e., a component that contains VRML). Each component has its own resources. Further, each component generates and owns its own object tree of static elements and inserts its tree into the object tree that is used to generate the page. Components therefore provide the ability to have multi-content documents, or Web pages. Multi-content documents are documents that are defined using multiple types of definitional constructs or syntax such as HTML, PDF, VRML, etc. Thus, a Web page can be generated by integrating multiple components independent of the definitional constructs used in each component.

As with a page, a component can have a set of resources associated with it. The resource set can include, for example, association(s), template(s), and/or custom logic. However, it not required that all three of these resources be associated with a component. A component can have one or none of these resources associated with it, for example.

Controller

A control object class, or controller, is used in the preferred embodiment to provide object cover for a component. Controller objects are dynamic elements that have associative behavior, or custom logic. A root control class is provided that is preferably an abstract class (i.e., objects are not instantiated from this class). The root control class defines data and behavior that is inherited by subclasses of the root class. The root control class encapsulates behavior used by a component. Such behavior includes the ability to: 1) associate with a corporate, or enterprise, object; 2) generate and own static element(s); and 3) insert static element(s) into an object tree.

In the preferred embodiment, the root control class contains subclasses. One subclass is for object classes that control HTML elements. This branch is referred to as the object control branch, or subclass. For instance, stringControl and imagecontrol object classes can be used to control the NSWHString and the NSWHImage object elements, respectively. A stringcontrol object instance instantiates an NSWHString static HTML element and inserts it in the template. A controller class is preferably provided for every element of the definitional syntax (e.g., every HTML element).

Another subclass of control objects is referred to as the scriptedControl class. This class is used to manage a page or a component of a page (a page can be comprised of one or more components). Referring to FIG. **6**A, for example, an instance of the scriptedControl object class, scriptedControl object **602**, owns the resources that include script **604**,

association **606**, and template **608**. A scriptedController (an instance of the scripted control class) is instantiated when a new page is created or when the page needs a component, for example. When a page is defined by script **604**, associations **606** and template **608**, scriptedControl object **602** is a control for the page. Alternatively, script **604**, associations **606** and template **608** define a portion of a page. In this case, scriptedControl object **602** is a control for a component.

The scriptedControl object **602** includes behavior to generate a tree of static elements such as object tree **624** in FIG. **6**B. A control object owns the tree of static HTML objects that it generates. Some or all of the objects in an object tree (e.g., Object tree **624** in FIG. **6**B) can be created programatically or instantiated from a pre-defined template (e.g., template **608**). For example, object tree **624** can be generated entirely at run-time. Alternatively, an initial definition of object tree **624** can be dynamically altered at runtime. Object tree **624** can be an object representation of a dynamic Web page. Alternatively, object tree **624** can be an object tree for a component of the Web page. The controller object owns its object tree.

The script **604**, associations **606**, and template **608** are used to generate all, or a portion of, a Web page. Script **604** contains some custom or scripted logic. Custom or scripted logic that is associated with a component and its control object can be implemented in any interpreted or compiled language. Script **604** is written in an objective C that is interpreted at runtime. However, script **604** can be written using any programming language (e.g., JAVA). Script **604** includes two sets of logic: "willSendWith" and "thanksUser". The "willSendWith" script is executed during Web page generation. It can be invoked by the logic associated with the previous page, for example. It contains a page-level persistence declaration for an employee object. In addition, it can contain logic to fetch an employee from the external data source and store the retrieved data in the employee object's properties (e.g., an employee name is retrieved and loaded into the "name" property of the employee object).

The "thanksUser" script is executed as an action taken once the Web page is returned by the client. It contains a local-level persistence declaration for "nextPage". It can also contain, for example, logic to determine the next page that should be generated and sent to the client as a response. In addition, the logic can create the new page object, request that the new page invoke its own logic, and return the identity of the next page to the application. The application can insert the identity of the next page in a response object (see a discussion of response object below in Event Processing).

A pre-defined template can be written using any Web page definitional language, syntax, or construct. For example, template **608** is written in HTML. Alternatively, a template can be written using JAVA, PDF, VRML, or ACROBAT, for example.

The particular syntax used to define an association file entry (i.e., an association). Association characteristics that are used in the preferred embodiment include: binding information, static element declarations, and property declarations. Property declarations can include instance variable initializations, for example.

Script **604**, associations file **606**, and template **608** are used to generate object graph **624** in FIG. **6**B. A root object **642** is instantiated as the root of the object tree. Root object **642** can be, for example, a group object. FORM **644** is a group object instantiated from template **608** (i.e., the NSTAG statement having a NAME property equal to "FORM") and association **606** (i.e., the association having the "FORM" label or name). String object **646**, form object **648**, and EMPLOYEE object **650** are included in FORM **644**. String **646** and form **648** are HTML static objects. EMPLOYEE object **650** is a group object that corresponds to the GROUP declarations in association **606** (i.e., the association having the "FORM.EMPLOYEE" label or name) and template **608** (i.e., the NSTAG statement having a NAME property equal to "EMPLOYEE").

Block **626** includes controller objects. The associations contained in the Associations file **606** and the statements in the template **608** include binding information for the controller objects. Control objects can be instantiated using the associations contained in association file **606**. Further, the associations contained in the Associations file **606** can be used by control objects to determine what static HTML elements should be created.

The form Control object **662** is bound to FORM **644** via the FORM association in the associations file **606** and FORM group template of template **608**. The formControl object **662** is instantiated from the object control subclass. It is used to manage a HTML static element. The formControl object **662** uses the information in the FORM association to instantiate a static element (e.g., NSWHForm static HTML element). Using the group identification ("FORM:") contained in the FORM association, the formControl object **662** instantiates form object **648** and inserts it in the object graph **624** inside the FORM group.

Associations can further be used to bind external data **630** to Web page **622** and its corresponding object graph **624**, controllers **626**, and context **628**. The inputControl **664** is instantiated from the "FORM.EMPLOYEE" association. It instantiates text object **654** and inserts it into the EMPLOYEE group **650**. The "FORM.EMPLOYEE" association contains a "value" property that is equated to "employee.name". The "value" property identifies the object that contains the external data retrieved from external data source **630**.

Each control object instance has an associated context. Associated enterprise objects are found within a scriptedControl's context. Preferably, each scriptedController has its own context object (i.e., an object that manages object pools). FIG. **10** provides examples of context objects. Controllers **1002** and **1022** are running in the same session and application. Controller **1002** has context object **1004**. Controller **1022** has context object **1024**.

Context object **1004** manages object pools **1006**, **1008**, and **1010**. Object pools **1006**, **1008**, and **1010** are unique to controller **1002**. Examples of object pools **1006**, **1008**, and **1010** are transaction, page and local object pools. Similarly, context object **1024** manages object pools **1026**, **1028**, and **1030** for its associated control object **1022**.

Object pools **1012** and **1014** are shared by controllers **1002** and **1020**. Examples of object pools **1012** and **1014** are application and session pools, respectively. Therefore, context objects **1004** and **1024** share in the management of object pools **1012** and **1014**.

Referring to FIG. **6**B, if employee object **672** does not already exist, the inputControl object **664** creates it in context **628** using its context object. Employee object **672** includes a "name" property. Script **604** contains a declaration for employee object **672** (i.e., "page employee;") that indicates that employee object **672** has page-level persistence.

Push and Pull Modes

Controller objects provide the ability to dynamically alter a page. For example, a controller can insert a static element into the template. The type and attributes of the static

elements created by the controller objects are a function of the control class. For example, an imageControl object instance generates an image HTML element. Further, a controller can modify the properties that are used to generate the page. In both cases, the controller causes the resulting Web page to be dynamically generated or altered at runtime.

In the present invention, corporate data can be accessed and used to generate a Web page. Data is retrieved from an external source (e.g., database) and inserted into a Web page. Input data returned via a Web page can be used to update the external data. A control object instance can be used to interface with the external data. A controller causes data retrieved from the external data source to be stored in the context. Data returned via a Web page is retrieved from the page and stored in a context.

FIG. **11**A provides an example of the push and pull modes used in the present invention. When a response is prepared, information (e.g., data or logic) is retrieved from an external data source (e.g., database **1104**) and pushed into a Web **1106**. This is referred to as push mode. When a request is being processed, information is pulled from a Web page **1106** and stored in the external data source **1104**. This is referred to as pull mode.

In the preferred embodiment, objects act as intermediaries between the Web page **1106** and the external data source **1104**. FIG. **11**B provides examples of intermediary objects.

In the push mode **1100**, information is pulled from the external data source **1104** into context objects **1108**. Information contained in context objects **1108** is used to update Web page definitional objects in object tree **1110**. When the Web page description (e.g., HTML) is generated using the objects in object tree **1100**, the data retrieved from external data source **1104** is pushed into the Web page **1106**. Therefore, in push mode, information is pulled from a data source and pushed into the Web page.

In pull mode, information is pulled from a Web page and pushed into storage. Referring to mode **1102**, information is pulled from Web page **1106**. For example, key-value pairs of objects in object tree **1110** contain the data pulled from Web page **1106**. Information in the key-value pairs is pushed into the context objects **1108**. Information contained in context objects **1108** is pushed into the data source **1104**.

Preferably, an interface such as NeXT's Enterprise Objects Framework is used to interface with an external database source. Use of enterprise objects as an interface with a database is more fully described in U.S. patent applications entitled Method of Mapping Objects to Multiple Tables in a Relational Database (Ser. No. 08/353,522), Method for Providing Stand-in Objects (Ser. No. 08/353,523), Dynamic Object Communication Protocol (Ser. No. 08/353,524), and Method for Associating Data Containing Objects With User Interface Objects (Ser. No. 08/353,525), filed on Dec. 7, 1994, assigned to the assignee of the present invention and incorporated herein by reference.

In the preferred embodiment, controllers are used to manage the push and pull modes, For example, a controller manages the pushing of data from an external source into a Web page. A controller manages the process of pulling information from a Web page. Data pulled from a Web page can be used to update an external source.

The inputControl **664** is an example of a controller that pushes and pulls data. The inputControl **664** has an instance variable (ivar) called value that is, for example, a variable object that points to employee object **672**. The variable object knows that it points to the "name" property of employee object **672**. The inputControl **644** can ask the variable object to set value in and get value from the "name" property of employee object **672**. Therefore, the variable object can be used as an intermediary between inputControl **644** and the corresponding corporate object, employee **672**. Alternatively, inputControl **644** could associate with employee object **672** using a pointer to employee object **672**.

In the push mode, a control object is responding to a "prepareResponse" message. In push mode, data is retrieved from the external data source and used to dynamically generate all or some portion of a page or component. The control object loads its associated script(s), association(s), and template(s). The control object manages the retrieval of data from the external data source and the modifications made to the object tree owned by the control object.

For example, inputControl **664** determines based on the "FORM.EMPLOYEE" association in associations **606** that a text HTML static element should be included in object tree **624**. It first looks at the object tree **624** to see if the text object **654** exists. If it does not exist, inputControl **664** creates text object **654** and inserts it into the object tree **624**. The inputControl object **664** sets the properties for text object **654** using the attributes or properties contained in the associations file **606**. The text object **654** includes a "value" property that can be used to initialize the input field **632**. The inputControl **664** uses the data contained in the "name" property of employee object **628** to set the "value" property of text object **654**.

In pull mode, a control object receives a "handleRequest" message. In pull mode, data is pulled from the Web page and can be pushed back to the external data source. The control object responds to the message by loading its associated script(s) and association(s). The control object can also respond to the message by taking data retrieved from a form returned by the client and storing it in a context object. Data stored in a context object can be used to update the external data source.

For example, inputControl **664** examines the request information to determine whether it contains a reference to it (e.g., name of the inputControl **664** in the request arguments). If it finds a reference, it will execute the appropriate behavior. For example, the client may have modified field **632** to specify a new name. The inputControl object **664** takes the value that is returned in the form and places it in the "name" property of employee object **672**. At that point, a data source object can be used, for example to update the database. There can be, for example, a method in the page object that sends a message to the data source object to update the database.

The movement of data in the push and pull modes is illustrated in FIG. **6**B using pathways **612**, **614**, **616**, and **618**. It should be understood that these are used for illustration purposes only. The pathway including **612**, **614**, and **616** provide one example of the movement of data from an external data source **630** to the Web page in push mode. For example, path **612** includes a data source object that can be used to fetch data from and store data in external data source **630**. Similarly, path **618** provides one example of the movement of data from the Web page to the external data source **630** in pull mode. A change to the name value field **632** in Web page **622** can be stored text object **654** and then moved to external data source **630** directly or via employee object **628**.

Data source **630** can be, for example, an enterprise's corporate database. Alternatively, data source **630** can be a electronic mail system or online services retrieval system, for example.

Repetition Control

The repetition control is a subclass of the control class. It is used to generate a repeating subcomponent of the Web page. A repeating subcomponent is a portion or block within the Web page that repeats (e.g., each half of Web page **334** in FIG. **3**B). To display each repetition at a client site, a block of HTML statements must be generated for each repetition. A repetition control object contains behavior that replicates the HTML that defines each repetition.

A repetition control object, or repetition controller, in conjunction with other logic and data is used to modify the object tree to accommodate the repeating subcomponent(s). For example, the HTML template in Table Four contains an ordered list HTML element. The content of the ordered list is not specified in the HTML template. In the preferred embodiment, the repetition controller creates a template to generate the contents of the ordered list. A repetition control object enumerates on an array to duplicate the template for each iteration. These objects generate a list of employees when the Web page document is generated.

FIG. **7**B provides the object tree of FIG. **7**A and an associated repetition controller and template. The association linked to the "GREETING" group specifies that an "NSWString" control object is to be instantiated. "NSWHString" object **712** is instantiated by the "NSWString" control object and inserted into the object tree as a child of the group whose name is contained in the name field of the association having the "NSWString" declaration. In the object tree, "NSWHString" object **712** is inserted under the "GREETING" group object **708**. The "NSWHString" object class provides cover for free-floating text. The property field in the association is used to assign a value to the "value" property of "NSWHString" object **712**. The "value" property of "NSWHString" object **712** contains the text string used to generate the HTML document.

The "WEBPEOPLE" group object **716** has a corresponding association. The association contains an "NSWRepetiton" control class reference. Based on this, an instance of the "NSWRepetition" control class is instantiated and is associated with "WEBPEOPLE" group object **716**. The "NSWRepetition" controller **718** provides behavior to generate repeating subcomponents of the Web page.

The "list" property field in the "WEBPEOPLE" association specifies a container object, "employees". The "employees" container object contains an array of objects that is used to populate the ordered list specified in the HTML template in Table Four. The repetition controller **718** enumerates on the "employees" container object. The "item" property field specifies a variable object that is used to update the context with specific employee information.

Template **740** is used as a template to generate repetition occurrences. Template **740** includes the "FN" group object **720** and "LN" group object **722**. "WEBPEOPLE" group object **742** is used to insert a repetition occurrence into the object tree under "WEBPEOPLE" group object **716**. "FN" group object **720** and "LN" group object **746** includes NSWHInput objects **744** and **746**, respectively.

The context of the application is used to store data that is used to modify the HTML objects contained in the duplicated template (e.g., input object **724**). The context is the name space of the application. Preferably, the context provides a data dictionary that contains the name of an object and a pointer to the object. When the "item" property is encountered in an association, an "NSWVariable" object is created and points to an "employee" object. Repetition controller **718** iterates on the items in the "employees" object array. Repetition controller **718** sends a "get_value" message to the employee name-value object. The variable object updates the context in response to the message.

When a "get_value" message is received by the "employee" object, it retrieves the next value of the array and stores it in its properties in the context. For example, the "employee" object can gets a first name and a last name of an employee from the "employees" array.

The "employee" object has behavior to respond to "set_value" message. The key-value pairs of a definitional object (e.g., HTML object) contains data that is retrieved and used to set a value in the "employee" object in response to a "set_value" message. When the "set_value" message is received by the "employee" object, it sets the context.

For each iteration, the repetition controller **718** duplicates template **740** (i.e., "FN" object **744**, "LN" object **746**, NSWHInput object **748**, and NSWHInput object **750**). The repetition controller **718** activates the declared objects in the duplicated template. Repetition controller **718** passes the HTML template and the information received in the client request (if any) to the declared objects.

When the "FN" group object **720** is activated, it modifies the objects contained in its group (i.e., NSWHInput object **724**). The value property of NSWHInput object **724** is modified using the "fn" property of the "employee" object (i.e., an employee's first name). The value property of NSWHInput object **726** is modified with the "ln" property of the "employee" object (i.e., an employee's last name). One iteration of the "employees" array is completed.

In the next iteration, the next index in the "employees" array is used. Repetition controller **718** duplicates the template, and modifies the objects in the duplicated template. The value property of NSWHInput duplicated object is modified using the new first name value (i.e., the value from the next object in the "employees" array) in the "fn" property of the "employee" object. The value property of NSWHInput duplicated object is modified with the new last name value in the "ln" property of the "employee" object. A second iteration of the "employees" array is completed. This process is repeated for each iteration of the array.

The repetition controller **718** inserts the trees for each repetition occurrence into the object tree. Repetition controller **718** uses the corresponding group designations to determine the location at which a template duplication is inserted into the tree. Referring to FIG. **7**C, repetition controller **718** adds "FN", "LN" NSWHInput and NSWHInput objects **720**, **722**, **724**, **726**, respectively, under "WEBPEOPLE" group object **716**. Similarly, "FN" "LN" objects **728** and **730**, and NSWHInput objects **732** and **734** are inserted into the tree under the "WEBPEOPLE" group object **716**.

In summary, according to the preferred embodiment, a repetition control generates a template either from scratch or using resources (e.g., HTML template, associations, etc.). The template is duplicated for each iteration or repetition occurrence. For each iteration, the repetition controller: 1) duplicates the template; 2) modifies the duplicated template; and 3) inserts the modified template into the object tree. Using this technique, it is only necessary to parse the definition to generate a template. The template can be used to generate each repetition.

Alternatively, the definition is parsed at the beginning of each iteration. The template is then modified to generate a particular occurrence of an iteration.

Subcontrols

A scriptedControl object can have subcontrols. A subcontrol can be another scriptedControl object. For example, a component has an associated scriptedControl object that is a

subcontrol of the page's scriptedControl object. A subcontrol can be an object control such as imageControl, stringControl, or inputControl. Subcontrols can be created programmatically or instantiated from a pre-defined declaration.

A subcontrol binds with a group element in the object tree of the subcontrol's parent control. FIG. **6**C provides examples of a parent tree and two component or sub-trees. Parent tree **682** contains static HTML element **684**. In addition, parent tree **682** contains two group elements (CAL group element **686** and IDEMP group element **688**). The parent control is a scripted control that has an associations file **696**.

Associations file **696** contains an IDEMP association. Using the IDEMP association, an IDEMP group element **688** is instantiated and inserted into the parent's tree **680**. CAL group element **686** is instantiated and inserted in tree **680** based on the CAL association in associations file **696**. Associations file **696** identifies that CAL and IDEMP are subcontrols of the page scriptedControl.

Object tree **624** in FIG. **6**B has been modified such that the root object **642** is an IDEMP group element. Association file **606** in FIG. **6**A is modified to include an IDEMP association. Similarly, object tree **690** has as its root object a CAL group element **692** based on its corresponding association file.

The IDEMP and CAL group elements in object tree **680** are place holders. They correspond to the tag values used in association file **696**. They further correspond to the tags used in the associations files for the CAL and IDEMP components. The parent's tags are matched with the subcontrol's tags to determine the locations at which the CAL and IDEMP component trees are to be inserted into object tree **680**.

Like the control class, a subcontrol generates one or more static elements. The type and attributes of the static elements created by a subcontrol are a function of the subcontrol class and the template that is passed to the subcontrol (e.g., passed as a group).

A subcontrol is an instance of a class derived from the root control class. Therefore, a subcontrol can have associative behavior that allows it to perform push and pull operations as described above. The associative behavior of a subcontrol vests in the scriptedControl as well. That is, a scriptedControl for a page can be associated with an enterprise object because one of its subcontrols is associated to that enterprise object.

The associative behavior described above can be extended to include subcontrols. For example, in push mode a control object can load its script(s), association(s) and template(s) and invoke its behavior. In addition, the control object transmits the "prepareResponse" message to its subcontrols. In pull mode, a control object forwards the "handleRequest" message to its subcontrol(s) in addition to performing the operations discussed above (i.e., load script, load associations, and invoke its behavior). Each subcontrol performs similar operations in response to the message that it receives.

Event Processing

A client uses a Web browser as a graphical interface to the Internet. The client can send a request (e.g., retrieve an HTML page) to an HTTP server using a Web browser. In response to a request, the client expects to receive something (e.g., a response) from the HTTP server. To respond to the request, the request is received and processed and a response is generated.

FIG. **9**A provides a block overview of components that can be used to handle a request. Browser **902** transmits a request to the WWW Server **904** (e.g., an HTTP server). WWW Server **904** transmits the request to CGI server **906**. CGI server **906** identifies the Web Object server **912** based on the information contained in the request, for example. Request information is transmitted from CGI server **906** to Web Object server **912**. Web Object **912** passes the request information to application **914**. A page **916** manages the generation of a response. Controllers **918**–**928** are used, for example, to generate the response.

The components illustrated in FIG. **9**A are used in the preferred embodiment of the present invention to process a request. It should be noted, however, that the transaction handling described herein can be performed with additional or fewer components. For example, CGI server **906** can transmit a request directly to application **914**. Event objects are used in the preferred embodiment to process a request. Event objects are, for example, page, application, session, request, and response objects.

A page object represents an HTTP response (e.g., an HTML page). A response is generated by a page object. A page object can be dynamic. Its representation can change. The representation can vary, for example, based on the state of the data to which the page object refers.

An application object manages page objects. The pages managed by a application object typically are related and can be combined to allow a client to accomplish a specific goal. For example, one page may need data that was entered in a previous, related page. Or, for example, a page may lead to another page. Pages can refer to the same data, for example.

A transaction is started by a client action. It ends when a response is received by a client. A request is the client action that initiates a transaction, for example. An HTML representation of a page is one example of a response that ends the transaction, for example. A transaction uses a request object and a response object. The request and response objects acts as a vehicle for passing information (e.g., invocation and response information). For example, request and response objects can be used to pass information between control objects. The application queries the information contained in the response and request object during event processing.

A response object is associated with a transaction. A response object retains information that is used to respond to the client. When a response is identified (e.g., next page to send to the client), the application builds a response object. The application sends a "prepareResponse" message to the response page and passes the information associated with the response to the response page. The response page manages the generation of the response.

A request object represents a user action that needs to be processed. The request object includes, for example, information regarding the application object (i.e., to handle the request), page object(s), and specific areas of a page that are selected or activated by the client. If a form is submitted by the client, the request includes data entered in the form, for example. The request can further contain state information including information regarding the pages previously traversed by the client and the data entered in these pages, for example.

A request preferably includes the following information: application name, pathname for desired page, session identification, transaction identification, sender identification, a digital signature, and form data. This information can be stored in the request object's properties. The application name identifies the application that handles the request. The pathname specifies the path, or location, for the

page in which the client took an action that initiated a request. Where the action is independent of a page (e.g., login panel), the pathname identifies the location of the page solicited by the client.

The session identification (ID) identifies the client's session. A transaction identification (ID) specifies the last transaction completed for the client for an existing session. A sender identification (ID) identifies the portion of a page that was acted upon to initiate the request. For example, when a request is initiated when the client clicks on an anchor in an HTML page, the Sender ID identifies the name of the anchor object in the corresponding page. A digital signature can be used as a key to perform verification. For example, the digital signature can be used to verify that a request is a valid request generated by a valid client. The digital signature can further be used to verify that the client is authorized to access the data of the specified session.

Form data is preferably a set of key-value pairs (e.g., each pair including a key and a value associated with the key). When a page submission generated the request, the key-value pairs contain client-entered data from the page that generated the request.

A session object represents a series of transactions. A session object exists for each transaction. Transactions that are related are preferably represented by the same session object. Transactions may be related in that they are handled by the same application, for example. When transactions are handled by the same application, an application can be said to manage a set of sessions, each of which manages a subset of the related transactions.

FIG. **9**B provides an illustration of the relationships that can exist between the event objects. Server **952** owns one or more applications such as application **954**. Application **954** owns one or more pages such as page **956** and a set of sessions such as session **958**. Response object **960** is aware of an instance of page object **956** (i.e., a response page). Request object is aware of an instance of page object **960** (i.e., a request page).

As described above, a request is transmitted to the Web Objects server from the CGI server. FIG. **5**A provides an example of a process flow executed by the Web Objects server to manage applications. A function of the Web Objects server is to dispatch a request to an application that processes the request.

At step **502**, the request is received by the Web Objects server. Using the information contained in the request, the Web Objects server identifies the application that will process the request. At step **504**, the Web Objects server determines whether the application is executing either on the same or a different server, for example. If the application is not currently loaded, the Web Objects server initiates the application at step **508** and processing continues at step **506**. If the application is executing, processing continues to step **506**.

At step **506**, the request is forwarded to the application. The Web Objects server receives a response from the application at step **508**. At step **510**, the response is forwarded to the client. The response is forwarded to the client via the CGI server **906**, the WWW Server **904**, for example. Processing ends at step **512**. Web Objects server is available to process each request that it receives in a manner such as described in FIG. **5**A.

An application processes a request that it receives from a Web Objects server. When a client request is received by the application, the application manages the update and retrieval of data from the context, processes the received form (i.e., the form received from the client that includes the client request), and processes any return form (i.e., a form sent to the client in response to the client request).

FIGS. **5**B–**5**D provides an Application Request Handling process flow. At step **522**, a request is received by the application. At step **524**, the application determines whether the request involves an existing session. For example, the application examines the contents of the session ID, transaction ID, sender ID and signature properties in the request object. If values for these properties are missing, the application proceeds to create a session to process the request at step **526**. Processing continues at step **534**.

If there are values for these properties, processing continues at step **528** to verify the request that requests access to an existing session. FIG. **5**E provides a Verify Request process flow. At step **562**, the application retrieves the session ID from the request information. At step **564**, the application determines whether the session ID is a valid session ID. That is, for example, the application determines whether the session contained in the request corresponds to a session ID for a session that is managed by the application. If the session ID is not a valid session ID, an error is raised at step **578** and processing returns to the Application Request Handling process (i.e., step **526** in FIG. **5**B).

If the application determines that the request's session ID is a valid session ID, processing continues at step **566**. At step **566**, the application retrieves the session using the session ID. At step **568**, the application retrieves a session key from the session. A session key is generated when a session is created. The session retains this value. The key is accessible by the session and the application that manages the session. To retrieve the session key, the application retrieves the session from the set of session that it manages. At step **572**, the application encrypts a string that contains request information (e.g., page path, session ID, previous transaction ID, and sender ID) is encrypted using an encryption algorithm. Any encryption algorithm can be used. Preferably, however, an encryption technique such as described in U.S. Pat. Nos. 5,159,632 and 5,271,061, entitled Method and Apparatus for Public Key Exchange in a Cryptographic System, assigned to the assignee of the present invention, and incorporated herein by reference. The Data Encryption Standard implemented in Unix can also be used with the present invention. It is preferable that the encryption scheme used:

1. does not deduce an input string from an output string;
2. does not deduce an encryption key from the input string and/or the output string; and
3. generates a large and unpredictable change in the output string when a small change (e.g., one character) occurs in the input string.

At step **574**, a portion of the encrypted string is examined against the request's digital signature. At step **576**, the application determines whether the encrypted string and the digital signature match. If a match does not exist, processing continues at step **578** to raise an error. Whether or not a match exists, processing returns to the Application Request Handling process (i.e., step **526** in FIG. **5**B).

When the encryption scheme described in U.S. Pat. Nos. 5,159,632 and 5,271,061 or Unix's Data Encryption standard is used, a match indicates that:

1. the digital signature was generated with knowledge of the session key; and
2. the page path, session ID, previous transaction ID, and sender ID values contained in the request were used to generate the digital signature.

Since the session key is known only to the application and the session, a match between the encrypted string and the

digital signature indicates that the request's information and the digital string were generated in a previous transaction of the session.

At step **524**, the application verifies that the request is a valid request. The application can use the information contained in the request object to verify the request, for example. The digital signature property contains information that identifies a session, transaction, sender, for example. If this is a new request, the digital signature does not contain a session ID or transaction ID.

After verifying the request (as described in FIG. **5**E, for example), the application continues to process the request as described in FIG. **5**B. At step **526**, the application determines whether the verification process identified a valid request. If it was not a valid request, processing continues at step **532** to raise an error and processing ends at step **556** of FIG. **5**D. The error condition can be transmitted back through the chain illustrated in FIG. **9**A and communicated to the client, for example.

Referring FIG. **5**B, the application has determined that the request is valid as well as the request's session. The application creates a transaction at step **534**. Initially, the transaction contains a reference to the request that resulted in its creation. The transaction tracks the state of the request handling.

Referring to FIG. **5**C, the application establishes context information for the transaction in step **536**. Context information includes the data that is created or modified during processing. Context information can include, for example, data created or modified by other transactions that exist in the same or different sessions. Preferably, the context is stored in objects that can be accessed by the transaction during its execution.

As described above, contexts are scoped. The data contained in a context is accessible by different sets of objects based on this scoping. Table Two contains examples of scoping. For example, application context objects are accessible by objects that are involved in any transactions of the application's session(s). Session context objects are accessible by any objects involved in any transactions of the current session. The scoping examples in Table Two and/or other scope designations can be used to specify the accessibility of context information.

The current transaction's application-level context is taken from the context maintained by the application. The transaction's session-level context is retrieved from the session object (e.g., the session identified in the request's session ID property). The transaction-level context is retrieved from the previous transaction's page-level context. Therefore, objects involved in the current transaction have access to the data from the previous transaction's context. The session manages the previous transaction's page context. The current transaction retrieves the previous transaction's page-level context from the session using the previous transaction ID specified in the request information. A context is associated with the transaction.

Referring to FIG. **5**C, the application determines a page for the request at step **538**. The application uses the path information contained in the request. If path information is not specified by the request, the application can use a default specification. The application instantiates a request object and a request page object at step **540**. The page object is, for example, an instance of the scriptedControl object class. Alternatively, a different object can be instantiated as the page object. For example, an object class can be associated with a given path. An instance of that object class is then instantiated based on the path information. The instantiated object should have the ability to respond to methods that are normally sent to a page object.

At step **542**, the application sends a "handleRequest" message to the page object. The message preferably includes a pointer to (or identification of) the request object. The page object responds to the "handleRequest" message. The page object extracts the information that it needs from the "handleRequest" message, for example. This information is contained in the request's form data or sender ID that are stored in the request object. The page object can use this information to determine what action needs to be taken in response to the client's request, for example. The page object can inform the transaction of what action is needed.

The page object forwards the "handleRequest" message to its children (e.g., subcontrols **918–928** of the page scriptedControl object **916** in FIG. **9**A) along with the request object. A subcontroller that receives the message examines the request object to determine whether it should be activated. If the subcontroller determines that it should be activated, it performs its operation(s) and can update the information contained in the request object.

Once all of the controllers have received the "handleRequest" message, the application looks at the request object to determine whether a controller has set an invocation. At step **546**, the application queries the request object to determine what action is needed. If an action is not needed, the page solicited by the request is used to generate the client request. Processing therefore continues at step **552**. If a controller has set an invocation, the application retrieves and activates the invocation at step **548**. The action returns a page to be used to generate the response to the client, for example.

At step **550**, the application creates a response page object. At step **552**, the page creates a root object for the tree or graph of objects. The objects contained in the graph are used to generate a Web page definition (e.g., a series of HTML statements). The application sends a "prepareResponse" message to the page object along with the response object at step **554**. Any method of passing information can be used with the present invention. The page object manages the generation of the object graph in response to the "prepareResponse" message. By sending this message, the application asks the page object to generate the object tree or graph of objects that contains HTML objects that can represent themselves in HTML.

The page object is, for example, a scriptedController. It determines whether it has any children that need to receive the "prepareResponse" message. If there are, it activates its children (e.g., subcontrols **918–928** of of the page scriptedControl object **916** in FIG. **9**A). The page object sends the "prepareResponse" message, the response object, and the group associated with its child. A similar messaging scheme is used by the subcontrols to activate their subcontrols. The result of this process is an object tree that can be used to generate an HTTP description (e.g., HTML definition of a Web page containing JAVA and VRML components).

At step **556**, the application sends a "generateDescription" message to the page object to generate the HTTP description for the client. The page object responds to the "generateDescription" message. The page object sends the message to the next level (its children) in the object tree.

The message is filtered down each level of the object tree in a similar manner (i.e., the parent level forwarding the message to the child level). An object in the object tree generates a portion of the Web page (e.g., HTML statements) in response to the "generateDescription" message. When the message is received and responded to by the leaf objects (i.e., lowest level) of the object tree, the Web

page's definition is complete. The application returns the response (e.g., a Web page definition) to the server for transmittal to the client in response to the client's request. The application's processing of the client request ends at step **560**.

Page Generation

When the object tree is completed, it contains the objects that are used to generate the ordered list in the HTML document. The HTML document is generated by sending a "create" message to the objects in the object tree. Referring to FIG. **7**C, a "create" message is sent to the page object **702**. The page object transmits a "create" message to the "SELECT_FORM" object **706**. The "create" message is sent to the children, grandchildren, etc. until it reaches the "WEBPEOPLE" object **716**. Because the "WEBPEOPLE" object **716** is a group object, it does not generate any HTML element statements. It passes the "create" message to its children (i.e., "FN" object **720**. "LN" object **722**, "FN" object **728**, and "LN" object **730**). The children objects send the "create" message to their children. Their children are input objects **724**, **726**, **732**, and **734**. Each of the input objects generates an INPUT HTML element statement using their property values (e.g., "type", and "value").

The response form can be generated, or rendered, at runtime using a previously-defined template, data stored in the context, and the form's objects. The response form can also be dynamically generated with or without an HTML template.

To render an HTML document, a message is sent to the page object by the application. The page object transmits a similar message to the root object. FIG. **8** provides a process flow for object-oriented HTML page rendering. At step **802**, the application sends a "generateDescription" message to the root object. The root object responds to the message by rendering itself and then sending the "create" message to its children. At step **804**, the root object identifies its children objects (in the object tree). At step **806**, the root object sends a "generateDescription" message to its children. At step **808**, a determination is made whether the children objects have children. If they do not, processing ends at step **810**. If they do, the message is sent to the children at step **806**. Processing continues until all levels of the object tree have received the "generateDescription" message.

Thus, the object-oriented approach used by the present invention provides the ability to dynamically generate Web page description. Some or all of a Web page description can be generated based on input received in a receive form, generated at runtime, or retrieved from an external data source (e.g., database or electronic mail system). A Web page description can be generated using resources such as template(s) declaration file(s), and custom logic. Alternatively, a Web page description can be generated from scratch without the aid of some or all of the resources.

Thus, a method and apparatus for developing and managing Internet transactions has been provided.

APPENDIX A

Class	Description
NSWHAnchor	Object cover for Anchor HTML element
NSWHCheckbox	Object cover for Input HTML element, type is "checkbox"
NSWHComment	Object cover for comments
NSWHElement	
NSWHForm	Object cover for Form HTML element
NSWHGroup	Cover for group extension. A group is a container for other objects.

APPENDIX A-continued

Class	Description
NSWHImage	Object cover for Image HTML element
NSWHImageInput	Object cover for Input HTML element, type is "image"
NSWHImagemapRegion	
NSWHInput	Abstract class for input HTML elements. All input HTML elements subclass this class
NSWHKeyValueCoding	Defines methods to maintain a dictionary of data values associated with keys. Values can be extracted from the dictionary to initialize an object, or retrieved from an object and stored in the dictionary.
NSWHList	A generic container class for HTML elements that can contain other elements or plain text. This class can be used for HTML elements that do not covered by another object class
NSWHListItem	Object cover for List Item HTML element
NSWHMarkedElement	
NSWHMenu	Object cover for Menu HTML element
NSWHOption	Object cover for Option HTML element
NSWHOrderedList	Object cover for Ordered List HTML element
NSWHPage	
NSWHRadio	Object cover for Input HTML element, type is "radio"
NSWHRectRegion	
NSWHRepetitionGroup	Object cover for repeating group HTML extension
NSWHReset	Object cover for Input HTML element, type is "reset"
NSWHSelect	Object cover for Select HTML element
NSWHString	Object cover for free-floating HTML text
NSWHSubmit	Object cover for Input HTML element, type is "submit"
NSWHText	Object cover for Input HTML element, type is "text"
NSWHTextArea	Object cover for TextArea HTML element
NSWHParser	
NSWHUnknownElement	Object cover for Unordered List HTML element
NSWHUnorderedList	Object cover for Input HTML element
NSWHUrl	Object cover for Uniform Resource Locator
NSWHUrn	Object cover for URN

What is claimed is:

1. A method of handling a Web page comprising the steps of:

receiving from a client a first returned page containing a request;

activating one or more event objects associated with said returned page, wherein at least one of said event objects has a programmably controllable persistence level associated therewith;

said event object capable of retention according to said programmably controllable persistence level for use with at least one second pages; and

processing said returned page using said event objects.

2. The method of claim **1** further comprising the steps of:

determining a response Web page;

creating a page object for said response Web page;

requesting said response Web page to invoke an associated script.

3. A method of handling a Web transaction comprising the steps of:

receiving from a client a first returned page containing a request;

activating event objects associated with said returned page, wherein at least one of said event objects has a programmably controllable persistence level associated therewith;

said event object capable of retention according to said programmably controllable persistence level for use with at least one second pages;

determining whether a response is needed;

performing the following when a response is needed:

(a) identifying a response page;

(b) activating event objects associated with said response page;

(c) generating said response;

(d) transmitting said response to said client.

4. The method of claim **3** further comprising the steps of:

verifying said request; and

terminating said request when said request is not valid.

5. The method of claim **4** wherein said verification step further comprises the steps of:

obtaining a first verification string associated with said request;

retrieving a session associated with said client;

retrieving a session key from said session;

generating a second verification string using said session key and a subset of the information contained in said request;

comparing said first verification string with said second verification string.

6. The method of claim **4** further comprising the step of setting state information.

7. The method of claim **3** wherein said step of activating event objects associated with said response page further comprises the steps of:

activating a plurality of control objects for managing the generation of said response page;

activating a response object for managing information about said response;

activating a plurality of context objects for maintaining state information.

8. The method of claim **7** wherein said step of generating said response further comprises the steps of:

creating a plurality of element objects;

transmitting a "generate" message to said plurality of control objects;

said control objects transmitting said "generate" message to said plurality of element objects:

said element objects generating a plurality of definitional statements for said response page.

9. The method of claim **3** wherein custom logic is associated with said response page, said step of performing further comprising the step of executing said custom logic if said plurality of control objects identify said custom logic.

10. An article of manufacture comprising:

a computer usable medium having computer readable program code embodied therein for controlling data contained in a Web page having an element definition comprising:

computer readable program code configured to cause a computer to associate an element of a first Web page with an object oriented object containing data for said element, wherein said object oriented object has a programmably controllable persistence level associated therewith;

said object oriented object capable of retention according to said programmably controllable persistence level for use with at least one second Web pages;

computer readable program code configured to cause a computer to retrieve data from an external data source into said object oriented object;

computer readable program code configured to cause a computer to generate said element definition for said element using said object oriented object data.

11. The article of manufacture of claim **10** wherein a declaration contains binding information that associates said element with said data object.

12. The article of manufacture of claim **11** wherein said element definition is updated when said Web page is displayed and input data is entered, said article of manufacture further comprising:

computer readable program code configured to cause a computer to retrieve input data from said updated definition;

computer readable program code configured to cause a computer to store said input data in said object oriented object;

computer readable program code configured to cause a computer to update said external data source with said object oriented object data.

13. An article of manufacture comprising:

a computer usable medium having computer readable program code embodied therein for processing a Web transaction comprising:

computer readable program code configured to cause a computer to receive a first request from a client;

computer readable program code configured to cause a computer to activate at least one object oriented object associated with a response to said request, wherein said object oriented object has a programmably controllable persistence level associated therewith;

said object oriented object capable of retention according to said programmably controllable persistence level for use with at least one second requests;

computer readable program code configured to cause a computer to generate using said at least one object oriented object at least one definition for an element of said response;

computer readable program code configured to cause a computer to transmit said at least one definition for an element to said client.

14. The article of manufacture of claim **13** wherein said element is an HTML element.

15. The article of manufacture of claim **14** wherein said computer readable code configured to cause a computer to generate at least one definition for an element further comprises:

computer readable program code configured to cause a computer to access a template for said at least one definition for an element;

computer readable program code configured to cause a computer to make a plurality of modifications to said template.

16. The article of manufacture of claim **15** wherein an external data source object contains data for said response, said computer readable program code configured to cause a computer to make a plurality of modifications to said template further comprises:

computer readable program code configured to cause a computer to retrieve data from said external data source object;

computer readable program code configured to cause a computer to include said data in said at least one definition for an element.

17. A method of handling a Web transaction comprising the steps of:

receiving from a client a first returned page containing a request;

activating one or more event objects associated with said returned page, wherein at least one of said event objects

has a programmably controllable persistence level associated therewith;

said event object capable of retention according to said programmably controllable persistence level for use with at least one second pages;

determnining whether a response is needed;

performing the following when a response is needed:

(a) identifying a response page;

(b) activating event objects associated with said response page;

(c) generating said response;

(d) transmitting said response to said client;

(e) verifying said request;

(f) terminating said request when said request is not valid;

(g) obtaining a first verification string associated with said request;

(h) retrieving a session associated with said client;

(i) retrieving a session key from said session;

(j) generating a second verification string using said session key and a subset of the information contained in said request; and

(k) comparing said first verification string with said second verification string.

18. The method of claim **17** further comprising the step of setting state information.

19. The method of claim **17** wherein said step of activating event objects associated with said response page further comprises the steps of:

activating a plurality of control objects for managing the generation of said response page;

activating a response object for managing information about said response;

activating a plurality of context objects for maintaining state information.

20. The method of claim **19** wherein said step of generating said response further comprises the steps of:

creating a plurality of element objects;

transmitting a "generate" message to said plurality of control objects;

said control objects transmitting said "generate" message to said plurality of element objects:

said element objects generating a plurality of definitional statements for said response page.

21. The method of claim **17** wherein custom logic is associated with said response page, said step of performing further comprising the step of executing said custom logic if said plurality of control objects identify said custom logic.

22. An article of manufacture comprising:

a computer usable medium having computer readable program code embodied therein for controlling data contained in a Web page having an element definition comprising:

computer readable program code configured to cause a computer to associate an element of a first Web page with an object containing data for said element, wherein said object has a programmably controllable persistence level associated therewith;

said object capable of retention according to said programmably controllable persistence level for use with at least one second Web pages;

computer readable program code configured to cause a computer to retrieve data from an external data source into said object;

computer readable program code configured to cause a computer to generate said element definition for said element using said object data;

wherein a declaration contains binding information that associates said element with said data object;

wherein said element definition is updated when said Web page is displayed and input data is entered, said article of manufacture further comprising:

computer readable program code configured to cause a computer to retrieve input data from said updated definition;

computer readable program code configured to cause a computer to store said input data in said object;

computer readable program code configured to cause a computer to update said external data source with said object data.

23. An article of manufacture comprising:

a computer usable medium having computer readable program code embodied therein for processing a Web transaction comprising:

computer readable program code configured to cause a computer to receive a first request from a client;

computer readable program code configured to cause a computer to activate at least one object associated with a response to said request, wherein said object has a programmably controllable persistence level associated therewith;

said event object capable of retention according to said programmably controllable persistence level for use with at least one second requests;

computer readable program code configured to cause a computer to generate using said at least one object at least one definition for an element of said response;

computer readable program code configured to cause a computer to transmit said at least one definition for an element to said client;

wherein said computer readable code configured to cause a computer to generate at least one definition for an element further comprises:

computer readable program code configured to cause a computer to access a template for said at least one definition for an element;

computer readable program code configured to cause a computer to make a plurality of modifications to said template.

24. The article of manufacture of claim **23** wherein said element is an HTML element.

25. The article of manufacture of claim **23** wherein an external data source object contains data for said response, said computer readable program code configured to cause a computer to make a plurality of modifications to said template further comprises:

computer readable program code configured to cause a computer to retrieve data from said external data source object;

computer readable program code configured to cause a computer to include said data in said at least one definition for an element.

* * * * *