(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 December 2004 (29.12.2004)

PCT

(10) International Publication Number
WO 2004/114093 A2

(51) International Patent Classification⁷: G06F

(21) International Application Number: PCT/US2004/019831

(22) International Filing Date: 21 June 2004 (21.06.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/479,952 20 June 2003 (20.06.2003) US
10/870,428 18 June 2004 (18.06.2004) US

(71) Applicant (*for all designated States except US*): THOMSON PROMETRIC [US/US]; 1000 Lancaster Street, Baltimore, MD 21202 (US).

(72) Inventors; and
(75) Inventors/Applicants (*for US only*): CROWHURST, Christopher; 1701 Whitehall Road, Whitehall, MD 21161. BOONE, Doug; 20721 York Road, Baltimore, MD 21202. KERSHAW, Roger, C.; 400 Beverly Court, Melbourne Beach, FL 32951.

(74) Agent: CHERDAK, Erik, B.; Arent Fox PLLC, 1050 Connecticut Avenue, NW, Suite 400, Washington, D.C. 200036 (US).

(81) Designated States (*unless otherwise indicated, for every kind of national protection available*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 2004/114093 A2

(54) Title: SYSTEM AND METHOD FOR COMPUTER BASED TESTING USING CACHE AND CACHEABLE OBJECTS TO EXPAND FUNCTIONALITY OF A TEST DRIVER APPLICATION

(57) Abstract: A system and method for computer-based testing includes a test driver that controls delivery of a computer-based test to one or more test candidates and that controls caching of test components during delivery of the test. The system includes various monitoring components, including monitoring of candidate progress, candidate performance, network bandwidth, network latency and server response, during delivery of the test and adjusting the source of the test components or the volume of the test components being cached for delivery of the test. Based upon this monitoring of the system, for example, if network communication failure is detected, the test candidate is able to continue computer-based testing while connectivity is being reestablished.

TITLE OF THE INVENTION

[0001] SYSTEM AND METHOD FOR COMPUTER BASED TESTING USING CACHE AND CACHEABLE OBJECTS TO EXPAND FUNCTIONALITY OF A TEST DRIVER APPLICATION

CROSS REFERENCE TO RELATED APPLICATIONS

[0002] This application is related to and claims priority from U.S. Provisional Patent Application No. 60/479,952, filed June 20, 2003, the disclosure of which is incorporated herein by reference in its entirety, and is further related to U.S. Patent Publication No. 20030203342, published on October 30, 2003 and entitled "METHOD AND SYSTEM FOR COMPUTER BASED TESTING USING CUSTOMIZABLE TEMPLATES", U.S. Patent Publication No. 20030196170, published on October 16, 2003 and entitled "METHOD AND SYSTEM FOR COMPUTER BASED TESTING USING A NON-DETERMINISTIC EXAM EXTENSIBLE LANGUAGE (XXL) PROTOCOL", U.S. Patent Publication No. 20030182602, published on September 25, 2003 and entitled "METHOD AND SYSTEM FOR COMPUTER BASED TESTING USING PLUGINS TO EXPAND FUNCTIONALITY OF A TEST DRIVER", and U.S. Patent Publication No. 20030138765, published on July 24, 2003 and entitled "METHOD AND SYSTEM FOR COMPUTER BASED TESTING USING AN AMALGAMATED RESOURCE FILE", and U.S. Patent Publication No. 20030129573, published on July 10, 2003 and entitled

"EXTENSIBLE EXAM LANGUAGE (XXL) PROTOCOL FOR COMPUTER BASED TESTING", all of which were filed concurrently and all of which are incorporated herein by reference in their entirety.

BACKGROUND OF THE INVENTION

Field of the Invention

[0003] The present invention is related to systems and methods used to facilitate computer based testing such as those that utilize network systems. More particularly, the present invention uses cacheable objects to expand functionality of a test driver application. And, even more particularly, such cacheable objects include cacheable data objects and cacheable program and application objects that may be used by a test driver application in facilitating test taking and administration.

Description of the Related Art

[0004] For many years, standardized testing has been a common method of assessing examinees as regards educational placement, skill evaluation, etc. Due to the prevalence and mass distribution of standardized tests, computer-based testing has emerged as a superior method for providing standardized tests, guaranteeing accurate scoring, and ensuring prompt return of test results to examinees.

[0005] Tests are developed based on the requirements and particulars of test developers. Typically, test developers employ psychometricians or statisticians and psychologists to determine the specific requirements specific to human assessment. These experts often have their own, unique ideas regarding how a test should be presented and regarding the necessary contents of that test, including the visual format of the test as well as the data content of the test. Therefore, a particular computer-based test has to be customized to fulfill the client's requirements.

[0006] FIG. 1 illustrates a prior art process for computerized test customization, denoted generally by reference numeral 10. First, a client details the desired test requirements and specifications, step 12. The computerized test publisher then creates the tools that allow the test publisher to author the items, presentations, etc., required to fulfill the requirements, step 14. The test publisher then writes an item viewer, which allows the test publisher to preview what is being authored, step 16.

[0007] An item presenter is then written to present the new item, for example, to the test driver, step 18. Presenting the new item to the test driver requires a modification of the test driver's executable code. The test driver must be modified so that it is aware of the new item and can communicate with the new item presenter, step 20. The test packager must then also be modified, step 22. The test packager, which may also be a compiler, takes what the test publisher has created and writes the result as new object codes for the new syntax. Subsequently, the scoring engine must also be modified to be able to score the new item type, step 24. Finally, the results processor must be modified to be able to accept the new results from the new item, step 26. This process requires no less than seven software creations or modifications to existing software.

[0008] U.S. Pat. No. 5,827,070 (Kershaw et al.) and U.S. Pat. No. 5,565,316 (Kershaw et al.) are incorporated herein by reference. The '070 and '316 patents, which have similar specifications, disclose a computer-based testing system comprising a test development system and a test delivery system. The test development system comprises a test document creation system for specifying the test contents, an item preparation system for computerizing each of the items in the test, a test preparation system for preparing a computerized test, and a test packaging system for combining all of the items and test components into a computerized test package. The computerized test package is then delivered to authorized examinees on a workstation by the test delivery system.

[0009] FIGS. 2A and 2B illustrate the test preparation process as disclosed in the '070 and '316 patents. Test developers assemble the test as shown at 32. As shown at 36, item selection is preferably automated (AIS) using the test development/document creation ("TD/DC") system or an equivalent test document creation system. Using "TD/DC", test developers enter the test specifications into the "TD/DC" system. Based on these specifications, "TD/DC" searches its central database for items, which satisfy the test specification, e.g., 50 math questions, 25 of which are algebra problems and 25, which are geometry problems. Then, the test developers review the items selected by "TD/DC" for sensitivity and overlap constraints described in the background section. If the test developer decides that the sensitivity or overlap constraints are not satisfied by the current selection of items, certain items may be designated to be replaced by another item from the database. In addition, test developers provide a test description specifying the directions, messages, timing of sections, number of sections of the test, etc. as shown at 42. If a computer adaptive test (CAT) is to be run, test developers may run a computer adaptive test simulation at 34, which are known to skilled test developers. Using the Test Preparation Tool (TPT) and TOOLBOOK 46, the test preparation system ("TPS") prepares the test level components as shown at 50. TOOLBOOK is commercially available from Asymetrix Corporation. The test level components include scripts 66, item table block sets 56, general information screens 58, direction screens 60, message screens 62, and tutorial units 64. Each of the test components will be described in detail below. As the components are prepared, the TPT stores them in a TPS network directory 52. Then, the components are entered into the TPS Production database 54. The components stored in the TPS Production database 54 will be retrieved during test packaging.

[0010] U.S. Pat. No. 5,513,994 (Kershaw et al.), which is incorporated herein by reference, discloses a centralized administrative system and method of administering standardized test to a plurality of examinees. The administrative system is implemented on a central administration workstation and at least one test workstation located in different rooms at a test center. The administrative system software, which provides substantially administrative functions, is executed from the central administration workstation. The administrative system software, which provides function carried out in connection with a test session, is executed from the testing workstations.

[0011] However, computer-based testing has expanded from standalone distribution administered at a local test center to wide area network distribution administered via clustered servers at multiple locations. Thus, a distributed computer-based testing system requires scalability to support continuous exam administration and a high volume of concurrent test candidates who may be located at many remote locations.

[0012] Additionally, computer-based tests have evolved from mere display of simple text-based content to include streaming of audio and video content. Thus, a distributed computer-based testing system demands sufficient system resources and storage capacity as well as efficient data communication management to serve bandwidth-intensive multimedia content in a consistent manner.

[0013] Moreover, computer-based test models have advanced to include adaptive and simulation test models. Thus, a distributed computer-based testing system must support a variety of complex test models.

[0014] Further, a distributed computer-based testing system must facilitate a fair testing environment within a dynamic networked environment to test candidates who may have varying workstation capabilities or network connectivity. A number of factors affect the creation and maintenance of a fair testing environment, including bandwidth mismatches and network latency between a test candidate workstation and a test distribution server as well as between a test distribution server and a test source server, the available system resources of the test source server, the test distribution servers and the test candidate workstations, and test component characteristics (e.g., whether the object is text, audio or video). Thus, it is necessary to monitor candidate progress, candidate performance, network bandwidth, network latency, and server response, among other testing environment variables, during computer-based testing and cache test components in response to changes in the testing environment in order to ensure timely and consistent delivery of the computer-based test. In other words, a distributed computer-based testing system must be adjustable to emulate a suitable testing environment on test candidate workstations concurrently executing the same computer-based test.

SUMMARY OF THE INVENTION

[0015] The present invention discloses a computer-based testing system that controls delivery of a computer-based test to a high volume of concurrent test candidates and that adapts delivery of the computer-based test in response to changes in the testing environment.

[0016] It is one feature and advantage of the present invention to deliver computer-based tests to a high volume of concurrent test candidates located at multiple locations.

[0017] It is another feature and advantage of the present invention to securely administer computer-based tests among concurrent test candidates located at multiple locations.

[0018] It is another feature and advantage of the present invention to monitor candidate progress during computer-based testing for ensuring that test components are timely available for delivery to a test candidate during testing.

[0019] It is another feature and advantage of the present invention to monitor candidate performance during computer-based testing for ensuring that suitable test components are available for delivery to a test candidate during testing, for example, to support Computer Adaptive Testing (CAT).

[0020] It is another feature and advantage of the present invention to monitor network bandwidth during computer-based testing for adapting delivery of a computer-based test to a test candidate in accordance with the network bandwidth.

[0021] It is another feature and advantage of the present invention to monitor network latency during computer-based testing for adapting delivery of a computer-based test to a test candidate in accordance with the network latency.

[0022] It is another feature and advantage of the present invention to monitor server response during computer-based testing for adapting delivery of a computer-based test to a test candidate in accordance with the server response.

[0023] It is another feature and advantage of the present invention to enable a test candidate to launch a computer-based test on the candidate's workstation prior to all the test components having been delivered to the candidate workstation.

[0024] It is another feature and advantage of the present invention to enable a test candidate to continue computer-based testing when network connectivity fails during computer-based testing.

[0025] These and other features and advantages of the present invention are achieved in systems and methods for computer-based testing including a test driver application that controls delivery of a computer-based test to a test candidate. Particularly, the system includes cacheable objects, including cacheable data objects (test items) and cacheable program and application objects (plugins), collectively, test components, to expand the functionality of the test driver application, enabling the test driver application to control caching of test components in response to changes in the testing environment during delivery of a computer-based test to a candidate workstation. The systems and methods include monitoring candidate progress, candidate performance, network bandwidth, network latency and server response during delivery of the computer-based test and adjusting either the source of test components or the volume of test components being cached for delivery of the test. Based upon such monitoring, for example, if network communication failure is detected, the test candidate is able to continue computer-based testing while connectivity is being reestablished in the background.

[0026] There has thus been outlined, rather broadly, the more important features of the invention and the preferred embodiments in order that the detailed description thereof that follows may be better understood, and in order that the present contribution to the art may be better appreciated. There are, of course, additional features of the invention that will be described hereinafter and which will form the subject matter of the claims appended hereto.

[0027] In this respect, before explaining the preferred embodiments of the invention in detail, it is to be understood that the invention is not limited in its application to the details of construction and to the arrangements of the components set forth in the following description or illustrated in the drawings. The invention is capable of other embodiments and of being practiced and carried out in various ways. Also, it is to be understood that the phraseology and terminology employed herein are for the purpose of description and should not be regarded as limiting.

[0028] As such, those skilled in the art will appreciate that the conception, upon which this disclosure is based, may readily be utilized as a basis for the designing of other structures, methods and systems for carrying out the several purposes of the present invention. It is important, therefore, that the claims be regarded as including such equivalent constructions insofar as they do not depart from the spirit and scope of the present invention.

[0029] Further, the purpose of the foregoing abstract is to enable the U.S. Patent and Trademark Office and the public generally, and especially the scientists, engineers and practitioners in the art who are not familiar with patent or legal terms or phraseology, to determine quickly from a cursory inspection the nature and essence of the technical disclosure of the application. The abstract is neither intended to define the invention of the application, which is measured by the claims, nor is it intended to be limiting as to the scope of the invention in any way.

[0030] These, together with other objects of the invention, along with the various features of novelty, which characterize the invention, are pointed out with particularity in the claims annexed to and forming a part of this disclosure. For a better understanding of the invention, its operating advantages and the specific objects attained by its uses, reference should be had to the accompanying drawings and descriptive matter in which there is illustrated preferred embodiments of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0031] FIG. 1 is a flow diagram of a prior art method of computerized test customization;

[0032] FIGS. 2A and 2B are block diagrams of a prior art process for test production;

[0033] FIG. 3 is a schematic diagram of a computer-based testing system;

[0034] FIG. 4 is a block diagram illustrating different types of plugins that are used with the computer-based testing system;

[0035] FIG. 5 illustrates various components that comprise an exam source file;

[0036] FIGS. 6A and 6B are a schematic illustrating the components, classes, and interfaces that comprise a test definition language compiler;

[0037] FIG. 7 is a schematic illustrating the components that comprise a test driver and a test administration system;

[0038] FIGS. 8A and 8B are schematics illustrating the classes and interfaces that comprise the test driver;

[0039] FIG. 9 illustrating the interfaces that comprise a structured storage;

[0040] FIGS. 10A and 10B are schematics illustrating the classes and interfaces that comprise the structure storage and associated operations;

[0041] FIG. 11 is a block diagram of main storage branches of an exam resource file;

[0042] FIG. 12 is a block diagram illustrating an exams branch of the exam resource file;

[0043] FIGS. 13A and 13B are block diagrams illustrating a forms branch of the exam resource file;

[0044] FIG. 14 is a block diagram illustrating an items branch of the exam resource file;

[0045] FIG. 15 is a block diagram illustrating a categories branch of the exam resource file;

[0046] FIG. 16 is a block diagram illustrating a templates branch of the exam resource file;

[0047] FIG. 17 is a block diagram illustrating a sections branch of the exam resource file;

[0048] FIG. 18 is a block diagram illustrating a groups branch of the exam resource file;

[0049] FIGS. 19A, 19B, 19C, and 19D are block diagrams illustrating an events sub-branch of the groups branch of the exam resource file;

[0050] FIG. 20 is a block diagram illustrating a plugins branch of the exam resource file;

[0051] FIG. 21 is a block diagram illustrating a data branch of the exam resource file;

[0052] FIG. 22 is a block diagram illustrating a formGroups branch of the exam resource file;

[0053] FIG. 23 is a block diagram illustrating an attributes branch of the exam resource file;

[0054] FIG. 24 is a block diagram illustrating a scripts branch of the exam resource file;

[0055] FIG. 25 is a block diagram illustrating a message box branch of the exam resource file;

[0056] FIGS. 26A, 26B, 26C, and 26D are block diagrams of an exam instance file;

[0057] FIG. 27 is a flow diagram of a method of computerized test customization;

[0058] FIG. 28 is a flow chart of a method of test production and test delivery;

[0059] FIG. 29 is a flow chart of a method for validation of test specification and content;

[0060] FIG. 30 is a flow chart of a method for test delivery;

[0061] FIG. 31 is a flow chart of a method of restarting a test after interruption;

[0062] FIG. 32 is a diagram of a life cycle of a plugin;

[0063] FIG. 33 is a flow diagram of a process for compiling plugins;

[0064] FIGS. 34A, 34B, 34C, and 34D are flow diagrams of a process for delivering plugins to an examinee during a computer-based test;

[0065] FIG. 35 is a block diagram illustrating an example of a network environment for a computer-based testing system according to the present invention;

[0066] FIG. 36 is a block diagram illustrating a caching architecture for a computer-based testing system according to the present invention;

[0067] FIGS. 37A and 37B are flow charts of methods for caching test components according to the present invention; and

[0068] FIGS. 38A, 38B, 38C, 38D and 38E are flow charts of methods for monitoring candidate performance, candidate progress, network latency, network bandwidth and server response according to the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0069] Reference now will be made in detail to the presently preferred embodiments of the invention. Such embodiments are provided by way of explanation of the invention, which is not intended to be limited thereto. In fact, those of ordinary skill in the art may appreciate upon reading the present specification and viewing the present drawings that various modifications and variations can be made.

[0070] For example, features illustrated or described as part of one embodiment can be used on other embodiments to yield a still further embodiment. Additionally, certain features may be interchanged with similar devices or features not mentioned yet which perform the same or similar functions. It is therefore intended that such modifications and variations are included within the totality of the present invention.

[0071] The present invention discloses a system and method of computer-based testing using a test driver that is, for example, object-oriented and is architected to dynamically add functionality through, for example, the use of an expansion module, and preferably through the use of plugins. The test driver preferably references component object model servers using standard interfaces, and uses, for example, class names (that can be an Active Document) defined in a custom test definition language entitled eXtensible eXam Language ("XXL") based on extensible Markup Language ("XML") format to interact with existing applications while offering the flexibility of allowing development of new plugins. These new plugins can be customized to a client's needs without changing the core test driver. The specific format and protocol of XXL is also described in U.S. Patent Publication No. 20030129573, published July 10, 2003 and entitled "EXTENSIBLE EXAM LANGUAGE (XXL) PROTOCOL FOR COMPUTER BASED TESTING", incorporated herein by reference.

[0072] The plugins advantageously enable the test driver to support, for example, new item types, navigation algorithms, information displays, scoring algorithms, timing algorithms, test unit selection algorithms, results persistence reporting, printed score reporting, and/or helm types without change to the test driver's executable. Plugins also allow expansion of the test driver's functionality without requiring the test driver to be recompiled or re-linked, and without requiring the test publisher to learn to program. Since plugins are written independently of the test driver, plugins can be written long after the test driver is built.

The client and the software developer can design and test the plugins and distribute the plugins to each test site. By using this method, large-scale regression testing of other examinations will not usually be necessary unless changes are made to the plugins that may be used by many examinations. The specific use of plugins is described in U.S. Patent Publication No. 20030182602, published September 25, 2003 and entitled "METHOD AND SYSTEM FOR COMPUTER BASED TESTING USING PLUGINS TO EXPAND FUNCTIONALITY OF A TEST DRIVER", incorporated herein by reference.

[0073] The test driver of the present invention controls delivery of a computer-based test to a test candidate, including controlling caching of test components during delivery of the test. In accordance to monitoring of the testing environment, including monitoring candidate progress, candidate performance, network bandwidth, network latency and server response, during delivery of the test, the test driver adjusts either the source of test components or the volume of test components being cached for delivery of the test. Based on such monitoring, for example, if network communication failure is detected, the test candidate is able to continue computer-based testing while connectivity is being reestablished in the background. By using this system, a uniform testing environment can be established and maintained during computer-based testing in a distributed computer-based testing environment.

[0074] I. Overview of Computer-Based Test Delivery System

[0075] FIG. 3 shows an overview of the software architecture for the computer-based test delivery system of the present invention, denoted generally by reference numeral 100. Test driver 110 is responsible for controlling all aspects of the computer-based test. Test driver 110 identifies candidates scheduled to take the computer-based test and identifies and creates the appropriate test. Test driver 110 then presents all of the test components to candidates using a display device (not shown), such as a computer monitor, and enables candidates to enter responses to test questions through the use of an input device (not shown), such as a keyboard, a mouse, etc. Test driver 110 also monitors the security of the test. For example, test driver 110 can prevent access to the Internet and can validate candidates, although, these functions are preferably performed by the test center administration system. Test driver 110 also monitors the timing of the test, providing relevant warnings to candidate regarding the elapsed time of the test and the time remaining for a particular section of the test or for the entire test. Test driver 110 is also responsible for scoring the test, once the test is completed or while the test is in progress, and for reporting the results of the test by physical printout using printer 182 or in a file format using candidate exam results file 180. If the test is interrupted while in progress, for example, due to a power failure, test driver 110 restarts the test, preferably at the point at which the test was interrupted, as will be described subsequently in more detail. Finally, if the test is left incomplete, test driver 110 cleans up the incomplete test. An incomplete test will have an exam instance file in the candidate's directory but will not have created a results file. A results file is created even though generally the candidate will fail. The number of items delivered to the candidate is recorded in the results file. Test driver 110 picks up where the event was interrupted and invisibly delivers the rest of the units of the test.

[0076] A test specification is authored by a test publisher according to the specifications of the client and stored in exam source files 130. Exam source files 130 include data files 132, XXL files 134, multimedia files 136, and hypertext markup language ("HTML") files 138. XXL files 134 include the test specification, which contains the client's requirements for the test, a bank of test items or questions, templates that determine the physical appearance of the test, plugins, and any additional data necessary to implement the test. Additional data is also stored in data files 132. For example an adaptive selection plugin may need a, b & c theta values. These values are stored in a binary file created by a statistical package.

[0077] HTML files 130 include, for example, any visual components of the test, such as the appearance of test items or questions, the appearance of presentations on the display device, the appearance of any client specified customizations, and/or the appearance of score reports. HTML files 130 preferably also include script, for example, VBscript and Jscript, or Java script. HTML files 130 are preferably authored using Microsoft's FrontPage 2000. FrontPage 2000 is preferably also used to manage the source files in a hierarchy that is chosen by the test publisher. Multimedia files 136 include, for example, any images (.jpg, .gif, etc.) and/or sound files (.mp3, .wav, .au, etc.) that are used during the test.

[0078] XXL compiler 140 retrieves XXL files 134 from exam source files 130 using interface 190 and compiles the XXL test content stored in XXL files 134. XXL compiler 140 stores the compiled test files in exam resource file 120. In another embodiment, exam source files 130 do not contain XXL files 134 and contains, for example, only multi-media files. In this embodiment, XXL compiler 140 is merely a test packager that writes the data directly to exam resource file 120 without modification or validation. The data appears in a stream under the "data" branch of exam resource file 120. The name of the stream is specified by the test author.

[0079] In a preferred embodiment, XXL files 134 also include XXL language that defines plugins 150, in which case, plugins 150 assist XXL compiler 140 in compiling XXL files 134. Test driver 110 preferably supports, for example, nine different types of plugins 150, including, for example: display plugin 152; helm plugin 154; item plugin 156; timer plugin 158; selection plugin 160; navigation plugin 162; scoring plugin 164; results plugin 166; and report plugin 168. Plugins 150, which are also included in XXL files 134, are the first XML files compiled into exam resource file 120.

[0080] Plugins 150 allow a test designer to customize the behavior of test driver 110 and are divided into two types, for example: visible plugins and invisible plugins, as shown in FIG. 4. The visible plugins, which include display plugin 152, helm plugin 154, and item plugin 156, enable the test driver to control what is presented visually to an candidate on the display device. The invisible plugins, which include timer plugin 158, selection plugin 160, navigation plugin 162, scoring plugin 164, results plugin 166, and report plugin 168, enable the test driver to control more functional aspects of the test. Plugins 150 are used to validate data stored in exam source files 130 that is to be used by one of plugins 150 during delivery of the test to the candidate, as is described below in greater detail. Plugins 150 are, preferably, component object model ("COM") objects, as described below. Plugins 150, may also utilize Java implementation. Plugins 150 are preferably written using Microsoft Visual C++ or Visual Basic 6.0 or any fully COM enabled language. Plugins 150 may be in or out-of-process, and, therefore, can exist as executable (".EXE") files or as dynamic link library (".DLL") files.

[0081] An application or component that uses objects provided by another component is called a client. Components are characterized by their location relative to clients. An out-of process component is an .exe file that runs in its own process, with its own thread of execution. Communication between a client and an out-of-process component is therefore called cross-process or out-of-process communication.

[0082] An in-process component, such as a .dll or .ocx file, runs in the same process as the client. It provides the fastest way of accessing objects, because property and method calls don't have to be marshaled across process boundaries. However, an in-process component must use the client's thread of execution.

[0083] Exam resource file 120 receives the compiled test content from XXL compiler 140 and plugins 150, if applicable, and stores the compiled test content in an object-linking and embedding ("OLE") structured storage format, called POLESS, which is described in greater detail below. Other storage formats may optionally be used. OLE allows different objects to write information into the same file, for example, embedding an Excel spreadsheet inside a Word document. OLE supports two types of structures, embedding and linking. In OLE embedding, the Word document of the example is a container application and the Excel spreadsheet is an embedded object. The container application contains a copy of the embedded object, and changes made to the embedded object affect only the container application.

In OLE linking, the Word document of the example is the container application and the Excel spreadsheet is a linked object. The container application contains a pointer to the linked object and any changes made to the linked object change the original linked object. Any other applications that link to the linked object are also updated. POLESS supports structured storage such that only one change made to an object stored in exam resource file 120 is globally effective. Test driver 110 comprises Active Document container application 112 for the visible plugins, display plugin 152, helm plugin 154, and item plugin 156, which function as embedded objects, preferably COM objects.

[0084] Both XXL compiler 140 and plugins 150 are involved in storing the compiled test content into exam resource file 120, if any of plugins 150 are being used. Exam resource file 120 comprises, for example, a hierarchical storage structure, as will be described in further detail below. Other storage structures may optionally be used. XXL compiler 140 determines to which storage location a specific segment of the compiled test content is to be stored. However, if any of plugins 150 are used to validate the portion of any of the data from exam source files 130, then the plugins 150 store the data directly to the exam resource file, based upon directions from XXL compiler 140. XXL compiler uses IPersistResource interface 192, co-located with I-Plugin interface 167 in FIG. 3, to control the persistence of the data to exam resource file 120. XXL compiler 140 and plugins 150 write the data to exam resource file 120 using POLESS interfaces 191.

[0085] FIG. 5 illustrates the contents of exam source file 130, which are compiled into exam resource file 120 by XXL compiler 140 and plugins 150. FrontPage 2000 Web 200 is used, for example, to author the test. Exam source files 130 contain media files 210, visual files 220, and logic files 230. Media files 210 are multimedia files used to enhance the presentation of the test, including, for example, XML data files 212, sound files 214, image files 216, and binary files 218. XML data files 212 include the XXL test definition language and the XXL extensions from the plugins 150 that use XML. The test specification, presentation, scoring and other information is specified in the XML files. Sound files 214 include any sounds that are to be used during the test, such as .mp3 files, .au files, etc. Image files 216 include any images to be used during the test, such as .jpg files, .gif files, etc. Binary files 218 include any data needed by a plugin 150 that is not in XXL format. Visual files 220 are HTML files that specify the visual presentation of the test as presented to the examine on the display device, including items files 222, presentation files 224, score report files 226, and custom look files 228. Items files 222 include HTML files that are used to specify the visual component of test questions, e.g., stems and distractors. Items files 222 are capable also of referencing external exhibits. An exhibit could be a chart, diagram or photograph. Formats of exhibits include, for example: .jpg, .png, etc. Presentation files 224 define what is seen by the candidate on the display device at a particular instant during the test. Score report files 226 is typically an HTML file with embedded script that includes, for example candidate demographics, appointment information, and candidate performance. The performance might include pass/fail, achievement in different content areas, etc. Custom look files 228 are typically HTML files with embedded script to layout, for example, the title bar and information contained therein. Logic files 230 are XML files that specify the functional aspects of the test, including test specification files 232, plugin files 234, item bank files 236, and template files 238. Test specification files 232 specify the content and progress of the test as provided by the client. Plugin files 234 define plugins 150 and contain any data necessary to implement plugins 150. Item bank files 236 include the data content and properties of the items, or test questions, that are to be presented to the candidate during the test. Properties of an item include the correct answer for the item, the weight given to the item, etc. Template files 238 define visual layouts that are used with the display screen during the test.

[0086] Referring again to FIG. 3, once a test has begun, test driver 110 accesses exam resource file 120 for the instructions and files needed to implement the test, using POLESS interfaces 193. Test driver 110 also accesses plugins 150 for additional data that expands the functionality of test driver 110 in the areas of items, navigation algorithms, information displays, scoring algorithms, timing algorithms, test unit selection algorithms, results persistence reporting, printed score reporting, and/or helm types. Test driver 110 communicates with plugins 150 using various COM interfaces 169. COM interfaces facilitate OLE linking. As stated previously, test driver 110 is an Active Document container application and plugins 150 are embedded objects. The COM interfaces function as communications paths between the container application and the objects.

[0087] There are, for example, ten COM interfaces utilized in computer-based test delivery system 100. IPlugin interface 167, which is also a COM interface, is supported by all of plugins 150. COM interfaces 169, therefore, includes the IPlugin interface. The IPlugin interface contains generic operations such as loading and unloading, required of all plugins 150. In addition to the global IPlugin interface, each plugin 150 also uses, for example, a second, individual COM interface 169 to communicate with test driver 110. Alternative structures of the IPlugin interface may also be used. Table 1 shows the relationship between each plugin 150 and the COM interface 169 used with that particular plugin 150.

TABLE 1: COM INTERFACE FOR PLUGINS.

PLUGIN	COM INTERFACE	DESCRIPTION
All Plugins 150	IPlugin	Passes data between the test driver and all plugins regarding generic operations, e.g., loading and unloading.

PLUGIN	COM INTERFACE	DESCRIPTION
Display 152	IDisplay	Passes data between the test driver and the visible plugins that handle title bars, displays, non-answered items, and summaries.
Helm 154	IHelm	Passes data between the test driver and the visible plugins that display navigation controls or reviews. Communicates with a navigation plugin to perform the actual navigation. Also functions as a user interface connection to the test driver.
Item 156	IItem	Passes data between the test driver and the visible plugins that govern test items or simulations.

PLUGIN	COM INTERFACE	DESCRIPTION
Timer 158	IUnitTimer	Passes data between the test drivers and the invisible plugins used to perform timing across examination sections.
Selection 160	ISelection	Passes data between the test drivers and the invisible plugins used to select forms, sections, groups, or items for delivery to the candidate.
Navigation 160	INavigate	Passes data between the test drivers and the invisible plugins used to control section navigation and define rules for traversing through the test.

PLUGIN	COM INTERFACE	DESCRIPTION
Scoring 164	IScore	Passes data between the test drivers and the invisible plugins used to control scoring of delivered testing units.
Results 166	IResults	Passes data between the test drivers and the invisible plugins that control writing of candidate results, for example, to candidate exam results file 180.
Report 168	IReport	Passes data between the test drivers and the invisible plugins that control printing of score reports and other material, for example, printed reference material and post exam instructions to printer 182.

[0088] Exam instance file 170 is used to restart a test if the test has been interrupted, for example, because of a power failure. During delivery of the test, exam instance file 170 receives examination state information from test driver 110 and plugins 150 regarding the state of all running objects being used to deliver the test. The examination state information includes the presentation that was being delivered on the display device before the interruption, the responses the candidate had entered in that presentation, etc. When the test is restarted, the exam instance file 170 loads the state information back to test driver 110 and plugins 150, allowing the test to return to operation at the point where the test had been interrupted. Preferably, the running state of all objects is saved to exam instance file 170 rather than of only some of the objects. Saving the state of only some of the objects to exam instance file 170 causes the potential problem of only a portion of the test information being restored after a test interruption. Exam instance file 170 may also store additional information relating to the test, including, for example: the timing utilized and time remaining on units of the exam, the current unit of delivery, candidate score, etc. Test driver 110 and plugins 150 communicate with exam instance file 170 using POLESS interfaces 195. Test driver

110 controls communications between test driver 110 and plugins 150 using IPersistInstance interface 196, which is collocated with COM interfaces 169 in FIG. 3.

[0089] Several administrative environments perform the administrative functions of computer-based test delivery system 100, for example: Test Center Manager ("TCM") Bridge 172; Educational Testing Service ("ETS") Bridge 174; and Unified Administration System ("UAS") 174. Administrative functions include, for example: checking-in an candidate, starting the test, aborting the test, pausing the test, resuming the test, and transmitting results.

[0090] There are preferably two ways to run test driver 110. The first is through a series of command line options and the second is using COM interfaces describing appointment information. The command line option exists for backwards compatibility in a standard ETS environment and a TCM environment. Table 2 shows a list of command line options test driver 110 supports. There are, for example, four programs which launch the test through the COM interface, for example: 1) LaunchTest.exe (for test production and client review); 2) UAS; 3) UTD2ETS.dll (an internal compatibility module for use with the ETS administration environment); and 4) UTD2TCM (for the Test Center Manger environment). Other number of environments and/or programs may optionally be used.

TABLE 2: COMMAND LINE OPTIONS SUPPORT BY TEST DRIVER.

SWITCH(ES)	OPTION(S)	PURPOSE
/? /help	n/a	Displays command line switches in dialog box.
/UnregServer	n/a	Unregisters the test driver core COM server.
/RegServer	n/a	Registers the test driver core COM server.
/T	Form name	Name of the form or form group to run in the exam.
/F	Resource file	The exam resource file to use.
/S	n/a	Suppress any printing.
/W	n/a	Run in close-of-day mode.
/TI	n/a	Set tracing level to information. (Very large instance file).

SWITCH(ES)	OPTION(S)	PURPOSE
/TW	n/a	Set tracing level to warning. (Large instance file.)
/TE	n/a	Set tracing level to error. (Average sized instance file.)
/K	Resource dir, SKSID, candidate director	Used to point to directories. A space separates each of the three options.

[0091] The administration environments use several interfaces to communicate with test driver 110. IAppointment interface 176 is part of UAS 174 and allows access by test driver 110 to candidate information for the candidate taking the test, such as demographics. The candidate information is included in candidate exam results file 180, which is created by the test driver. ILaunch2 interface 177 functions as the primary control interface for UAS 174 and allows UAS 174 to control various components such as test driver 110, screen resolution change, accommodations for disabled candidates, candidate check-in, etc., in a test center, which is the physical location where the candidate is taking the test. ITransfer interface 199 transfers candidate exam results file 180 and other files back to UAS 174. IPrint interface 198 sends information regarding any reports to printer 182.

[0092] The test driver is described in greater detail in U.S. Patent Publication No. 20030182602, entitled "METHOD AND SYSTEM FOR COMPUTER BASED TESTING USING PLUGINS TO EXPAND FUNCTIONALITY OF A TEST DRIVER", incorporated herein by reference.

[0092] II. XXL Compiler Interfaces and Classes

[0093] FIGS. 6A and 6B illustrate the main diagram for XXL compiler 140. XXL compiler 140 comprises the following classes, for example: cCompile 2000; cData 2004; cArea 2006; cTemplate 2008; cCategory 2010; cItem 2012; cPresentation 2014; cGroup 2016; cSection 2018; cForm 2020; cFromGroup 2022; cExam 2024; cMsgBox 2026; cChecksum 2028; cEvent 2030; cResult 2032; cReport 2024; cPlugin 2036; and cXXL 2038.

[0094] The main interface to XXL compiler 140 is ICompile interface 2002. ICompile interface 2002 is implemented by cCompiler class 2000. All control and initiation of compilation of exam source files 130 into exam resource file 120 occurs by way of this single public interface. The core, non-plugin related elements of the XXL test definition language, as stored in XXL files 134, are compiled by classes in XXL compiler 140. For example, cSection class 2018, compiles the section element, and cGroup class 2016 compiles the group element.

[0095] ICompile interface 2002 supports the following operations, for example: createResource( ); addSource( ); addData( ); closeResource( ); about( ); linkResource( ); openResource( ) and getCryptoObject( ). CreateResource( ) creates a resource file, for example, an XXL based resource file such as exam resource file 120. AddSource( ) compiles an XXL file into the resource file. AddData( ) adds a file directly to a data branch of the resource file. CloseResource( ) closes the resource file. LinkResource( ) links a resource in the resource file and is performed after all compiling of the source files are completed. GetCryptoObject( ) returns an ICrypto object containing the current encryption setting of POLESS, as described below.

[0096] The classes of XXL compiler 1040, e.g., cForm 2020 and cItem 2012, handle individual XXL core language elements. All of these classes compile the specific XXL source element into exam resource file 120. All of these class language elements are also symbols used in later references. Therefore, the classes all derive from cSymbol class 2040. cSymbol class 2040 allows the classes of XXL compiler 140 to reside in a symbol table.

[0097] For example, the XXL element plugin 150 appears as follows in XXL files 134:

```
3 <plugin name="helmNextPrevious"
progid="UTDP.cNextPrevious" />
```

[0098] This XXL call causes an instance of cPlugin class 2036 to be created, compiles the source, and writes the compiled result to exam resource file 120. The name and ID of Plugin 150 is also added to the symbol table for later reference.

[0099] XXL compiler 140 also contains the following token classes, for example: cToken 2042; cTokenCreatorNoRef 2044; cTokenCreator 2046; ctokenCreatorRef 2048; cTokenCreatorBase 2050; and cTokenFactory 2054. These token classes are involved in the identification of tokens. Tokens turn into symbols after identification. Symbols are any class derived from cSymbol, e.g., cTemplate, cSection, etc.

[0100] XXL compiler 140 also contains the following symbol table classes, for example: cPluginSymbolTable 2058; cTemplateSymbolTable 2060; cSymbolTable 2062; cFFGSymbolTable 2064; cSGPSymbolTable 2066; and cSymbolTableBase 2068. These classes are varieties of symbol tables. There are different symbol tables for different groups of symbols. A group of symbols define a name space for the symbol. Common symbol table functions are located in the base symbol table classes and templates.

[0101] All content and specification destined for a plugin 150 appears in the data element in XXL. For example, below is an item definition in XXL:

4 <item name="wantABreak1" skipAllowed="false"> <data> <multiChoice correctAnswer="A" maxResponses="1" minResponses="1" autoPrompt="false" URI="itembank/info_item.htm#wantABreak"/>- ; </data> </item>

[0102] The item element is handled by a cItem class 2012 object. The data element in the XXL definition is handled by a cData class 2004 object. Item plugin 156 Plugin 150 will receive the source to compile from the cData class 2004 object, in this example, a multiChoice element.

[0103] cWrapXML class 2052, a wrapper class for XML DOM nodes, supports error handling. cCustomAttributes class 2056 compiles the custom attributes XXL element. cWrapPropertySet class 2070 is a wrapper class for a POLESS property storage.

[0104] III. Test Driver Interfaces and Classes

[0105] A. Interfaces

[0106] FIG. 7 shows test driver 110, UAS 174, and the interfaces used by and between the test driver 110 and UAS 174 to deliver the test. UAS 174 defines ILaunch2 interface 177, which is used by UAS 174 to initiate test events. ILaunch2 interface 177 is an extension of ILaunch interface 178, which, in other embodiments of the present invention, is also used by UAS 174 to initiate test events. UAS 174 also defines and implements additional interfaces, for example: IAppointment interface 176; IPrint interface 198; and ITransfer interface 199. IAppointment interface 176 transfers examinee candidate information and appointment details from UAS 174 to test driver 110, as is illustrated by the dashed arrow connecting IAppointment interface 176 to test driver 110. IPrint interface 198 allows UAS 174 to send print requests to printer 198 regarding reports, for example, score reports. ITransfer interface 199 allows UAS 174 to request the transfer of information from candidate exam results file 180 back to UAS 174.

[0107] Test driver 110 defines various interfaces to allow test driver 110 to communicate with different parts of computer-based test delivery system 100. Test driver 110 includes, for example, ten COM interfaces 169 to communicate and transfer data with plugins 150. (See Table 1 above) The COM interfaces 169 are denoted in FIG. 7 as follows, for example: IDisplay interface 169a; IHelm interface 169b; IItem interface 169c; IUnitTimer interface 169d; ISelection interface 169e; INavigate interface 169f; IScore interface 169g; IResults interface 169h; IReport interface 169i; and IPlugin interface 169j.

[0108] Test driver 110 and plugins 150 communicate and transfer data with exam resource file 120 using, for example, three IPersistResource interfaces 192: IPersistResourceStream interface 192a; IPersistResourceSet interface 192b; and IPersistResourceStore interface 192. IPersistResource interfaces 192 are used by plugins 150 during compilation of exam source files 130 and are used by both test driver 110 and plugins 150 during delivery of the test. During compilation of exam source files 130, XXL compiler 140 directs plugins 150 in which storage location of exam resource file 120 to store any information that plugins 150 have validated. Plugins 150 can then retrieve the stored information from exam resource file 150 during delivery of the test. Other number of interfaces and different combination of functionality may alternatively be used.

[0109] Information is saved from plugins 150, or from XXL compiler 140 in general, to exam resource file 120, for example, as either a stream of data, as a set of data, or as a storage structure, depending on which of the three IPersistResource interfaces 192 is implemented to save the information from plugins 150, to exam resource file 120. IPersistResourceStream interface 192a saves the information, for example, as a stream of data or other data storage format. A stream of data is simply a stream of bytes stored as a linear sequence. IPersistResourceSet interface 192b saves the information, for example, as a set of data. A set of data is preferably a name-value property pair. For example, the name of a particular property for an item is distractors and the value is the number of distractors required for that item. IPersistResourceSet interface 192 allows the name-value property pair to be saved together in exam resource file 120. IPersistResourceStore interface 192c saves the information, for example, in a directory format with storage areas. The directory format allows other streams of data to be saved within the storage area, other property sets to be stored within the storage area, and for sub-storages to be saved under the storage area.

[0110] IPersistInstance interface 196, likewise, comprises, for example, three, different interfaces, for example: IPersistInstanceStream interface 196a; IPersistInstanceSet interface 196b; and IPersistInstanceStore interface 196c. Examination state information is saved to exam instance file 170 as, for example, a stream of data, as a set of data, or as a storage element, depending on which of the three IPersistResource interfaces 192 is implemented.

[0111] Two of the interfaces, IContainerNotify interface 200 and IContainerNotifyHelm interface 206, function as callback interfaces from plugins 150 to test driver 110.

IContainerNotify interface 200 allows a visible plugin to inform test driver 110, for example, that the plugin is displayed and ready for examinee interaction. IContainerNotifyHelm interface 206 allows helm plugin 154 to request navigation from test driver 110 after receiving an input from the examinee to move to another section of the test. IMore interface 202 is used to convey whether the examinee has seen all content in a presentation. For example, a "more" button appears in place of the next button when the content exceeds the window length. When the examinee scrolls to the bottom, the "more" button disappears and is replaced with the "next" button. Collection interface 204 is used by test driver 110 to hold any group entities, for example, categories and sections of the test.

[0112] The remaining interfaces are, for example, Microsoft defined Active Document interfaces, used to implement OLE linking functions of test driver 110 and the visible plugins, display plugin 152, helm plugin 154, and item plugin 156. IOleInPlaceFrame interface 210 controls the container's top-level frame window, which involves allowing the container to insert its menu group into the composite menu, install the composite menu into the appropriate window frame, and remove the container's menu elements from the composite menu. IOleInPlaceFrame interface 210 sets and displays status text relevant to the end-place object. IOleInPlaceFrame interface 210 also enables or disables the frames modeless dialogue boxes, and translates accelerator key strokes intended for the container's frame. IOleInPlaceUI window interface 211 is implemented by container applications and used by object applications to negotiate boarder space on the document or frame window. The container provides a RECT structure in which the object can place toolbars and other similar controls, determine if tools can in fact be installed around the objects' window frame, allocates space for the boarder, and establishes a communication channel between the object and each frame and document window. IAdviseSync interface 212 enables containers and other objects to receive notifications of data changes, view changes, and compound-document changes occurring in objects of interest. Container applications, for example, require such notifications to keep cached presentations of their linked and embedded objects up-to-date.

[0113] Calls to IAdviseSync interface 212 methods are a synchronous, so the call is sent and then the next instruction is executed without waiting for the calls return. IOleWindow interface 213 provides methods that allow an application to obtain the handle to the various windows that participate in-place activation, and also to enter and exit context-sensitive help mode. IOleInPlaceSite interface 214 manages interaction between the container and the objects in-place client site. The client site is the display site for embedded objects, and provides position and conceptual information about the object. IOleClientSite interface 215 is the primary means by which an embedded object obtains information about the location and extent of its display site, its moniker, its user interface, and other resources provided by its container. Test driver 110 called IOleClientSite interface 215 to request services from the container. A container must provide one instance of IOleClientSite interface 215 for every compound-document it contains. IOleDocumentSite interface 216 enables a document that has been implemented as a document object to bypass the normal activation sequence for in-place-active objects and to directly instruct its client site to activate it as a document object. A client site with this ability is called a "document site".

[0114] B. Core Classes

[0115] FIGS. 8A and 8B illustrate the main classes of test driver 110 and the interfaces between test driver 110 and plugins 150. Also shown are the classes that interface to UAS 174. ITransfer interface 199, IPrint interface 198, ILaunch2 interface 177, and IAppointment interface 176 represent the connections from test driver 110 to UAS 174, as described previously. Some of the lines depicted in FIG. 8 are solid and some are dashed. The solid lines, for example, between IcResults interface 240 and cEvent class 252, represent inheritance. The dashed lines, for example, between IExam interface 222 and IPlugin interface 169j, represent instantiation.

[0116] Inheritance, or generalization, relates to a generalized relationship between classes that shows that the subclass shares the structure or behavior defined in one or more superclasses. A generalized relationship is a solid line with an arrowhead pointing to the superclass. Instantiation, or dependency, represents a relationship between two classes, or between a class and an interface, to show that the client class depends on the supplier class/interface to provide certain services. The arrowhead points to the supplier class/interface. Some services from a supplier class to a client class include, for example: the client class access a value (constant or variable) defined in the supplier class/interface; methods of the line class invoke methods of the supplier class/interface; and methods of the client class have signatures whose return class or arguments are instances of the supplier class/interface. For instantiation, the cardinality of the relationship is illustrated in FIG. 8 if the relationship represents containment. Cardinality specifies how many instances of one class may be associated with a single instance of another class. Cardinality can be shown for relationships to indicate the number of links allowed between one instance of a class and the instances of another class.

[0117] Test driver 110 also has several interfaces and implementing classes. Test driver 110 interfaces include, for example: IExam interface 222; IMsgBox interface 224; ICategory interface 232; IForm interface 238; IcResults interface 240; IcReport interface 242; IScript interface 246; ISection interface 250; IPresentation interface 248; and/or IcItem interface 256. The classes that implement the main interfaces include, for example: cScreenMinimum class 226; cFormGroup class 228; cPlugin class 230; cArea class 234; cTemplate class 236; cActivePlugin class 250; and cEvent class 252. The interfaces that are prefaced by "Ic" have names that already exist for plugins 150 to enact, for example, item plugin 156 implements IItem interface 169c. IcItem interface 256, however, is the interface implemented by test driver 110 class cItem (not shown). Of course, any number of interfaces may be used, depending on the necessary functionality.

[0118] The core class cExam (not shown) implements ILaunch2 interface 177 so that UAS 174 can control test driver 110. The appointment object, which implements IAppointment interface 176, is the main object UAS 174 supplies to test driver 110. The appointment object is available to plugins 150 by way of IPlugin interface 169j. Furthermore, all plugins 150 get (IExam) using the IPlugin interface 169, also.

[0119] The cExam class selects and delivers the form, using cFormGroup class 228 and IForm interface 238. The form delivers results using IcResults interface 240, reports using IcReport interface 242, and sections contained with in the test using ISection interface 250. Classes that are in the test delivery chain preferably derive from cEvent class 252.

[0120] The cResults class (not shown) delivers a results plugin 166 that implements IResult interface 169i. The cReport class (not shown) delivers a report plugin 168 that implements IReport interface 169h. The cSection, cGroup, and cForm classes (not shown) use several invisible plugins 150 to control the delivery of the test. These plugins 150 are timer plugins 158, which implement IUnitTimer interface 169d, selection plugins 160, which implement ISelection interface 169e, scoring plugins 164, which implement IScore interface 169g, and navigation plugins 162, which implement INavigate interface 169f. The cPresentation class (not shown) supplies data to its template for the display of the presentation. The three visible plugins 150 are created and controlled through cTemplate class 236 and child objects cArea class 234. Item plugins 156 have an extension class in the cItem class (not shown) that wraps the item plugin 156 and provides generic extended services that all item plugins 156 implements. The cItem class in test driver 110 is a wrapper class. The cItem class provides two base services, for example: generic item functionality and access to item plugin 156, which is the wrapping function. Item generic functionality includes, for example: having an item name, having an item title, determining if the item is scored or un-scored, determining whether the item has been presented to the examinee, etc. These services are generic to all items and are provided by test driver 110. Item plugins 156 perform the actual scoring of the item, which is unique to each item type. Item plugins 156 present the content of the item and allow the examinee to interact with the item. These services are unique to each item type.

[0121] In addition to the interfaces described previously, test driver 110 implements IRegistry interface 220, which allows VB code to access the Windows registry. Test driver 110 also implements ILegacyItem interface 258 and ILegacyScore interface 260, which are defined by test driver 110 and are implements by certain item plugins 156 and scoring plugins 164. ILegacyItem interface 258 and ILegacyScore interface 260 allow old item types that existed in previous test drivers to report results like the previous test drivers. For some tests, test driver 110 must report results for old item types, which had very specific ways of reporting results. ILegacyItem interface 258 and ILegacyScore interface 260 allow the new item plugins 156 that represent old item types to report this legacy format of information to result plugins 166 trying to imitate previous test drivers.

[0122] A complete description of test driver 110 classes and interfaces is included in Appendix A.

[0123] IV. POLESS

[0124] All persistent storages, exam resource file 120 and exam instance file 170, preferably utilize POLESS. POLESS allows data to be embedded, linked, or references as external files from the persistent storage to test driver 110 and Active Document container application 112 (FIG. 3). POLESS supports a hierarchical tree structure with node or branch level additions, replacements, and deletions. POLESS also supports optional data encryption at the node level. The type of encryption employed depends on the destination of the information in the persistent storage. For example, different encryption keys may optionally be used for data being routed to test centers, data being routed to administrative data centers, and data being routed for client use (e.g., client review). Microsoft Crypto-API is preferably used to perform encryption of data in the persistent storage. Finally, POLESS also supports optional compression at the node level, preferably using Lempal-Zev compression.

[0125] POLESS is an extension of OLE structured storage compound document implementation. A compound document is a single document that contains a combination of data structures such as text, graphics, spreadsheets, sound and video clips. The document may embed the additional data types or reference external files by pointers of some kind. There are several benefits to structured storage. Structured storage provides file and data persistence by treating a single file as a structured collection of objects known as storage elements and streams. Another benefit is incremental access. If test driver 110 or plugins 150 need access to an object within a compound file, only that particular object need be loaded and saved, rather than the entire file. Additionally, structure storage supports transaction processing. Test driver 110 or plugins 150 can read or write to compound files in transacted mode, where changes made can subsequently be committed or reverted.

[0126] A. POLESS Components

[0127] FIG. 9 shows the major components that support POLESS and the interfaces that connect the components. POLESS 300 may be either exam resource file 120 or exam instance file 170. POLESS 300 utilizes PKware library component 330 for storage compression and decompression. POLESS 300 uses Crypto API component 332, a Microsoft application, for storage encryption and decryption. Crypto API component 332 relies on a crypto service provided ("CSP") 334 to perform the actual encryption algorithms. Access to the services of these components is facilitated by standard (API) interfaces exposed by these components.

[0128] OLE2SS component 310 contains all the interface definition that makeup structure storage. These interfaces can be realized by any structured storage implementation, such as compound document implementation OLE2 320 and POLESS 300. The interfaces include, for example: IStream interface 340; ISequentialStream interface 342; IStorage interface 344; and IRootstorage interface 346. POLESS 300 additionally implements IStreamVB interface 348 and IStorageVB interface 350.

[0129] IStreamVB interface 348 supports several functions, for example: ReadVB( ); WriteVB( ); Clear( ); Reset( ); get_sName( ); get_oStream( ); and CopyTo( ). ReadVB( ) reads a specified number of bytes to a data array. WriteVB( ) writes the byte data to the stream. Clear( ) clears the stream of all data. Reset( ) sets position to the beginning of the stream. get_sName( ) is a read-only function that returns the name of the stream. get_oStream( ) is a read-only function that returns the IStream interface 348. CopyTo( ) copies a source stream to a destination stream.

[0130] IStorageVB interface 350 supports several functions, for example: Clear( ); CommitVB( ); RevertVB( ); sElementName( ); bStorage( ); oElement( ); CreateStream( ); OpenStream( ); CreateStorage( ); OpenStorage( ); get_sName( ); get_oStorage( ); get_nCount( ); GetCompression( ); GetEncryption( ); GetCRC( ); CreateStreamLinked( ); CreatePropertyStg( ); OpenPropertyStg( ); SetClass( ); RegisterAlias( ); Destroy( ); and get_ElementType( ). Clear( ) clears the storage of all elements. CommitVB( ) causes transacted mode changes to be reflected in the parent. RevertVB( ) discards changes made since the last commit. sElementName( ) returns the name of the element. bStorage( ) returns TRUE if the element is a sub-storage. oElement( ) returns IStreamVB interface 348 or IStorage interface VB 350 for the element. CreateStream( ) creates and opens a stream and returns IStreamVB interface 348.

[0131] OpenStream( ) opens a stream and returns IStreamVB interface 348. CreateStorage( ) creates and opens a nested storage and returns IStreamVB interface 348. OpenStorage( ) opens an existing storage and returns IStreamVB interface 348. get_sName( ) is a read-only function that returns the name of the storage. get_oStorage( ) is a read-only function that returns IStorage interface 350. get_nCount( ) is a read-only function that returns a count of the elements. GetCompression( ) returns the status of file compression. GetEncryption( ) returns the status of file encryption. GetCRC( ) returns the status of file CRC checking. CreateStreamLinked( ) creates and opens a linked stream and returns IStreamVB interface 348. CreatePropertyStg( ) creates and opens a property storage and returns IPropertyStorageVB interface 414. OpenPropertyStg( ) opens a property storage and returns IPropertyStorageVB interface 414. SetClass( ) sets the CLSID for the storage. RegisterAlias( ) registers a pluggable protocol. Destroy( ) destroys the specified elements. get_ElementType( ) is a read-only function that returns the type of the element.

[0132] B. POLESS Classes

[0133] FIGS. 10A and 10B illustrate the main class of POLESS 300, the interfaces used to implement the classes, and the flow of the creation of streams 424 and storages 426.

cFileRoot class 400 is the first object instantiated and is used to create a new or open an existing a POLESS file. cStorageRoot class 406 is returned, which is a slightly overloaded version of cStorage class 410. From cStorageRoot class 406 creates or opens cStream class 408 and cStorage class 410, from which any streams or storages and sub-storages of those can be created or opened, respectively. For instance, cStorage class 410 creates cPropertyStorage class 412, which creates storage for property sets. The classes implement interfaces that perform operations and/or define attributes that further define the function or properties of the class. A complete description of POLESS 300 classes and interfaces is included in Appendix B.

[0134] 1) cFileRoot Class

[0135] cFileRoot class 400 is the root POLESS class and controls the creation and opening of all POLESS files. cFileRoot class 400 is generally instantiated first before any other POLESS objects can be created, although other sequences are possible. cFileRoot class 400 implements IFileRoot interface 401, which is collocated in FIG. 10 with cFileRoot class 400. IFileRoot interface 401 is used to open one file at a time and is not released until all other storage object 426, stream object 424, and property storage interfaces are released and the file is ready to be closed. cFileRoot class 400 and IRoot interface support the following operations, for example: StorageFileCreate( ); StorageFileOpen( ); CryptoGet( ); bStorageFile( ); StorageAmalgamatedGet( ); DeltaFileCreate( ); DeltaFileApply( ); GetObjectFromPath( ); CreateStreamFromBSTR( ); MemoryStreamFromStream( ); GetPicture( ); and SavePicture( ).

[0136] StorageFileCreate( ) creates a new storage file, returns the root storage to interface, marks the new structured storage file as a POLESS file by storing the class ID ("CLSID") of this class in a stream in the root storage. StorageFileOpen( ) opens an existing storage file and returns the root storage interface. CryptoGet( ) gets a default configured crypto class and should be set and used on the open or create of the storage file. bStorageFile( ) returns true if the file provided is an OLLE structured storage file and not a POLESS storage file. StorageAmalgamatedGet( ) gets an empty small cStorageAmalgamated class 404. DeltaFileCreate( ) creates a POLESS difference file by comparing the original POLESS file to the updated POLESS file. DeltaFileApply( ) applies a

POLESS delta file and applies the original POLESS file to the delta file to create an updated POLESS file. GetObjectFromPath( ) uses monikers to retrieve the object named by the path and returns a pointer to the object retrieved. CreateStreamFromFile( ) creates a structured storage stream and populates it with the contents of the file. CreateStreamFromBSTR( ) creates a structures storage stream and fills it with the specified string. MemoryStreamFromStream( ) is used to copy a stream to a newly created memory stream object. GetPicture( ) loads a picture from stream object 424. SavePicture( ) saves the picture into the stream 426.

[0137] 2) cCrypto Class

[0138] cCrypto class 402 controls the configuration of the encryption/decryption of POLESS 300. cCrypto class 402 has the following attributes, for example: sProviderName; eProviderType; sContainerName; and sPassword. SProviderName represents the name of CSP 334 being used to perform the encryption/decryption services. eProviderType is the type of CSP 334. The field of cryptography is large and growing. There are many different standard data formats and protocols. These are generally organized into groups or families, each of which has its own set of data formats and way of doing things. Even if two families used the same algorithm, for example, the RC2 block cipher, they would often use different padding schemes, different key links, and different default modes. Crypto API is designed so that a CSP provider type represents a particular family. sContainerName is the key name and must be provided by cCrypto class 402. sPassword is an optional password on the public/private key pair and can only be entered by a human operator. The password can be used for review disks and their resource files.

[0139] cCrypto class 402 implements ICrypto interface 401 and they support the following properties and method, for example: ProviderName; Password; FileType; Algorithm; EnumProviders( ); and EnumAlgorithms( ). Get_ProviderName( ) returns the name of the Crypto provider. Put_ProviderName( ) sets the name of the Crypto provider. Get_Password( ) and Put_Password( ) are only used for sponsor resource files. Get_FileType( ) gets the file type and put_FileType( ) sets the file type. Get_Algorithm( ) gets the encryption algorithm and put_Algorithm( ) sets the encryption algorithm. EnumProviders( ) returns an enumerator for the list of installed providers. EnumAlgorithms( ) enumerate a list of algorithms for the current provider.

[0140] 3) cStorageAmalgamated Class

[0141] cStorageAmalgamated class 404 is an implementation of IStorage interface 344. cStorageAmalgamated class 404 holds references to an ordered collection of IStorage objects. When a stream is opened, cStorageAmagalmated class 404 searches the collection of storage objects in order to find the first storage object that has the requested stream and returns this stream. cStorageAmalgamated class 404 handles compound storage resolution and delegates all other work to cStorage class 410. cStorageAmalgamated class 404 is, for example, read-only. cStorageAmalgamated class 404 will not allow stream or storages to be created but is primarily for reading exam resource file 120. cStorageAmalgamated class 404 implements IStorageAmalgamated interface 405. cStorageAmalgamated class 404 and IStorageAmalgamated interface 405 support the following operations, for example: StorageAdd( ); ClearStorage( ); OpenStorageAmalgamated( ); and OpenPropertyStgAmalgamated( ). StorageAdd( ) adds a new storage to the collection of storages. ClearStorage( ) clears all the storage objects from the collection.

OpenStorageAmalgamated( ) opens a sub-storage of the current amalgamated storages in an amalgamated fashion. OpenPropertyStgAmalgamated( ) opens a property storage of the current amalgamated storages in an amalgamated fashion. Amalgamation is described in greater detail, in U.S. Patent Publication No. 20030129573, entitled "EXTENSIBLE EXAM LANGUAGE (XXL) PROTOCOL FOR COMPUTER BASED TESTING," incorporated herein by reference.

[0142] 4) cStorageRoot Class

[0143] cStorageRoot class 406 is the POLESS implementation of IStorage interface 344 and IRootstorage interface 346. cStorageRoot class 406 handles any storage object 426 that is POLESS specific and then delegates work to the cStorage class 410. IRootstorage interface 346 supports the SwitchToFile( ) operation, which copies the current file associated with the storage object to a new file, which is then used for the storage object and any uncommitted changes. cStorageRoot class 406 also implements IPersistFile interface 418, which provides methods that permit an object to be loaded from or saved to a disk file, rather than a storage object or stream. Because the information needed to open a file varies greatly from one application to another, the implementation of IPersistFile::Load on the object preferably also open its disk file. IPersistFile interface 418 inherits its definition from IPersist, so all implementations must also include the GetClassID( ) method of IPersist interface 418.

[0144] 5) cStream Class

[0145] cStream class 408 is the POLESS implementation of IStream interface 340. cStream class 408 handles any storage object 426 that is POLESS specific and then delegates work to compound document implementation OLE2 320. The specific work includes compression/decompression and encryption/decryption of stream object 424.

[0146] IStream interface 340 supports the following operations, for example: Seek( ); SetSize( ); CopyTo( ); Commit( ); Revert( ); LockRegion( ); UnlockRegion( ); Stat( ); and Clone( ). Seek( ) changes the seek pointer to a new location relative to the beginning of stream object 424, the end of stream object 424, or the current seek pointer. SetSize( ) changes the size of stream object 424. CopyTo( ) Copies a specified number of bytes from the current seek pointer in stream object 424 to the current seek pointer in another stream object 424. Commit( ) ensures that any changes made to a stream object 424 open in transacted mode are reflected in the parent storage object. Revert( ) discards all changes that have been made to a transacted stream since the last call to IStream::Commit. LockRegion( ) restricts access to a specified range of bytes in stream object 424. Supporting this functionality is optional since some file systems do not provide this operation. UnlockRegion( ) removes the access restriction on a range of bytes previously restricted with IStream::LockRegion. Stat( ) retrieves the STATSTG structure for the stream object 424. Clone( ) creates a new stream object that references the same bytes as the original stream but provides a separate seek pointer to those bytes.

[0147] IStreamVB interface 348 is an automation friendly version of IStream interface 340. IStreamVB interface 348 supports the following operations, for example: Read( ); Write( ); Clear( ); Reset( ); get_sName( ); get_oStream; and CopyTo( ). Read( ) reads data from stream object 424. Write( ) writes data, including the entire byte array, to stream object 424. Clear( ) clears stream object 424 of all data. Reset( ) resets the position in stream object 424 to the beginning of stream object 424. Get_sName( ) returns the name of the stream. Get_oStream( ) returns the IDispatch interface. CopyTo( ) copies the contents of a source stream to a destination stream.

[0148] 6) cStorage Class

[0149] cStorage class 410 is the POLESS implementation of IStorage interface 344 and IcStorage interface 411. cStorage class 410 handles any storage object 426 that is POLESS specific and then delegates work to compound document implementation OLE2 320.

[0150] IStorage interface 344 supports the following operations, for example: CreateStream( ) ; OpenStream( ); CreateStorage( ); OpenStorage( ); CopyTo( ); MoveElementTo( ); Commit( ); Revert( ); EnumElements( ); DestroyElement( ); RenameElement( ); SetElementTimes( ); SetClass( ); SetStateBits( ); and Stat( ). CreateStream( ) creates and opens a stream object 424 with the specified name contained in a storage object. OpenStream( ) opens an existing stream object 424 within a storage object using specified access permissions. CreateStorage( ) creates and opens a new stream object 424 within a storage object. OpenStorage( ) opens an existing storage object 426 with the specified name according to the specified access mode. CopyTo( ) copies the entire contents of an open storage object 426 into another storage object. The layout of the destination storage object may differ from the layout of the source storage object. MoveElementTo( ) copies or moves a sub-storage or stream object 424 from one storage object 426 to another storage object.

[0151] Commit( ) reflects changes for a transacted storage object 426 to the parent level. Revert( ) discards all changes that have been made to the storage object 426 since the last IStorage::Commit operation. EnumElements( ) returns an enumerator object that can be used to enumerate storage objects 426 and stream objects 424 contained within a storage object. DestroyElement( ) removes the specified storage object 426 or stream object 424 from a storage object. RenameElement( ) renames the specified storage object 426 or stream object 424 in a storage object. SetElementTimes( ) sets the modification, access, and creation times of the indicated storage element, if supported by the underlying file system. SetClass( ) assigns the specified CLSID to a storage object. SetStateBits( ) stores state information in a storage object, for example up to 32 bits. Stat( ) returns the STATSTG structure for an open storage object.

[0152] IStorageVB interface 350 is an automation friendly version of IStorage interface 344. IStorageVB interface 350 supports the following operations, for example: Clear( ); Commit( ); Revert( ); sElementName( ); bStorage( ); bElement( ); CreateStream( ); OpenStream( ); Createstorage( ); OpenStorage( ); get_sName( ); getoStorage( ); get_nCount( ); GetCompression( ); GetEncryption( ); GetCRC( ); CreateStreamLinked( ); CreatePropertyStg( ); OpenPropertyStg( ); SetClass( ); RegisterAlias( ); Destroy( ); and get_ElementType( ). Clear( ) clears the storage of all elements, e.g. sub-storages and streams. Commit( ) ensures that any changes made to a storage object opened in transacted mode are reflected in the parent storage. For non-root storage objects in direct mode, this method has no effect. For a root storage, it reflects the changes in the actual device, for example, a file on disk. For a root storage object open in direct mode, the commit( ) method is always called prior to releasing the object. Commit( ) flushes all memory buffers to the disk for a root storage in direct mode and will return an error code upon failure. Although releasing the object also flushes memory buffers to disk, it has no capacity to return any error codes upon failure. Therefore, calling releasing without first calling commit( ) causes indeterminate results. Revert( ) discards all changes that have been made to the storage object since the last Commit( ) operation.

[0153] sElement( ) returns the name of the element. bStorage( ) returns true if the element is a sub-storage. bElement( ) returns either iStreamVB interface 412 or iStreamVB interface 414 or IStorageVB interface 412 for the selected element. CreateStream( ) creates and opens a stream object with the specified name contained in the storage object. Nothing is returned if the stream cannot be created. OpenStream( ) opens an existing stream object within this storage object in the specified access mode. Nothing is returned if the stream cannot be opened. CreateStorage( ) creates and opens a new storage object nested within the storage object. Nothing is returned if the storage cannot be created. OpenStorage( ) opens an existing storage object with a specified name in the specified access mode. Nothing is returned if the storage cannot be opened. Get_sName( ) returns the name of the storage. Get_oStorage( ) returns the IDispatch interface, which exposes objects, methods and properties to programming tools and other applications that support Automation. COM components implement the IDispatch interface to enable access by Automation clients, such as Visual Basic.

[0154] Get_nCount( ) returns the count of elements in the storage. GetCompression( ) determines if streams may be compressed in the file and if enabled streams may optionally be compressed when created. GetCRC( ) indicates whether a cyclic-redundancy-check ("CRC"), or a digital signature, check is to be performed on the file. CreateStreamLinked( ) creates a link to a stream in another POLESS file. CreatePropertyStg( ) creates a property storage. OpenPropertyStg( ) opens a property storage. SetClass( ) assigns the specified CLSID to a storage object. RegisterAlias( ) registers an alias to a storage in the POLESS file for access by the pluggable protocol. Destroy( ) destroys the specified element. Get_ElementType( ) is a read-only command that returns the type of the element.

[0155] 7) cPropertyStorage Class

[0156] cPropertyStorage class 412 implements IPropertyStorage interface 413, which supports the following operations, for example: ReadMultiple( ); WriteMultiple( );

DeleteMultiple( ); ReadPropertyNames( ); WritePropertyNames( ); DeletePropertyNames( ); SetClass( ); Commit( ); Revert( ); Enum( ); Stat( ); and SetTimes( ). ReadMultiple( ) reads property values in a property set. WriteMultiple( ) writes property values in a property set. DeleteMultiple( ) deletes property values in a property set. ReadPropertyNames( ) gets corresponding strung names fro given property identifiers. WritePropertyNames( ) creates or changes string names corresponding to given property identifiers. DeletePropertyNames( ) deletes string names for given property identifiers. SetClass( ) assigns a CLSID to a property set. Commit( ) flushes or commits changes to a property storage object, as is done with the command IStorage::Commit, described previously. Revert( ) discards all changes made since the last commit call when a property storage is opened in transacted mode. Enum( ) creates and gets a pointer to an enumerator for properties within a property set. Stat( ) receives statistics about a property set. SetTimes( ) sets modification, creation, and access times for a property set.

[0157] IPropertyStorageVB interface 414 is an automation friendly version of IPropertyStorage interface 413 that manages the persistent properties of a single property set.

IPropertyStorageVB interface 414 supports the following operations, for example: ReadVB( ); WriteVB( ); Delete( ); CommitVB( ); RevertVB( ); SetClass( ); get_nCount( ); CopyTo( ); GetName( ); WriteMultiple( ); and ReadMultiple( ). ReadVB( ) reads the value of a specified property from the property set. WriteVB( ) writes a value for a specified property to the property set. If the property does not exist the property/value pair will be created. If the property already exists, the value will be updated if opened in eAccess_Write mode. Delete( ) removes a property from the property set. CommitVB( ) flushes or commits changes to a property storage object, as is done with the command IStorage::Commit, described previously. RevertVB( ) discards all changes made since the last commit call when a property storage is opened in transacted mode. SetClass( ) assigns the specified CLSID to a property storage object. Get_nCount( ) returns the count of properties in the property set. CopyTo( ) copies the contents of the source property set to a destination property set. GetName( ) returns the name of the specified property. WriteMultiple( ) writes property values in a property set. ReadMultiple( ) reads property values in a property set.

[0158] 8) cPropertyStorageAmalgamated Class

[0159] cPropertyStorageAmalgamated class 416 implements IPropertyStorageAmalgamated interface 417, which supports the following operations, for example: PropertyStorageAdd( ) and ClearStorage( ). PropertyStorageAdd( ) adds a property set to the collection of property sets. ClearStorage( ) clears the collection of property sets.

[0160] C. POLESS Exam Resource File

[0161] FIGS. 11 and 12-25 illustrate the POLESS layout of exam resource file 120. Exam resource file 120 stores the various pieces of compiled information from exam source files 130, as shown in FIG. 5. Exam resource file 120 contains all of the content required to deliver the test. However, where the test is media-intense, exam resource file 120 will contain the core elements for the test with "links" to the external content. XXL compiler 140 and plugins 150 store the compiled information to exam instance file 120 using one of IPersistResourceStream interface 192a, IPersistResourceSet interface 192b, or IPersistResourceStore interface 192 to store the compiled information as a stream of data, a set of data, or a storage element, respectively. In a preferred embodiment, the layout of exam resource file

120 is in a hierarchical POLESS format that directly implements the format of the XXL test definition language. The test developer uses the XXL test definition language to create the logic files 230 and data files 212 (FIG. 5) of exam source file 130. By having a storage structure that follows the format of the XXL test definition language, the incremental access aspect of POLESS is easily implemented. XXL compiler 140 determines the storage location in exam resource file 120 that stores a particular piece of compiled information, even information stored into exam resource file 120 by one of plugins 150.

[0162] FIG. 11 illustrates the main storage branches of exam resource file 120, which corresponds to the top-level elements of the XXL test definition language, denoted by reference numeral 500. The main storage branches of exam resource file 120 are, for example: exams branch 550; forms branch 600; items branch 650; category branch 700; templates branch 750; sections branch 800; groups branch 850; plugins branch 900; data branch 950; formGroups branch 1000; attributes branch 1050; scripts branch 1100; and message box ("Msgbox") branch 1150. Other storage branches may alternatively be used.

[0163] Exam branch 550, as seen in FIG. 12, stores, for example, the primary attributes, properties, and data that govern the test. Exam branch 550 can store information for various tests, as is denoted by the three, vertical ellipses. A specific test is identified by the data stored in name attribute storage 552. Again, the various tests may each be identified by a different name, as denoted by the solid border around name attribute storage 552 or other identification scheme. Attributes storage 554 stores, for example, version information 555, and title information 556 of the test as a stream of data or other data storage format. Title information 556 is optional, as is denoted by the broken border. Any optional, customized information regarding the test is stored in custom properties 558 as a property storage or other data storage format. Information relating to the forms of the test are optionally stored in forms property storage 560. A form is a fixed or substantially fixed order of testing events. Many different forms can be stored in forms storage 560, giving flexibility to test driver 110 in controlling progression of the test. FormGroups storage 562 optionally stores information relating to a collection of exam forms as a stream of data or other data storage format. Preferably, a single form from the formGroup is chosen to deliver to an examinee. The selection of the form from the group is performed by a selection plugin 160. Exam branch 550 preferably contains at least one forms storage 560 either independently or within formGroups storage 562. Other information relating to the test may be stored under exam branch 550. Other storage formats may optionally be used

[0164] Forms branch 600, as seen in FIGS. 13A and 13B, stores, for example, the primary attributes, properties, and data that govern the progress of the test. Forms branch 600 can store information for various forms, as is denoted by the three, vertical ellipses. As described previously, a form is a fixed or substantially fixed or substantially fixed order of testing events. A single form is identified by the data stored in name attribute storage 602. Other identification formats may optionally be used. Again, the various forms may each be identified, for example, by a different name, as denoted by the solid border around name attribute storage 602. Attribute storage 604 stores, for example, begin section information 605, end section information 606, event information 607, and optionally stores version information 608, title information 609, skip allowed information 610, restartable information 611, with information 612, height information 613, and bit depth information 614. All information stored in attribute storage 604 is stored as a stream of data or other data storage format. Begin section information 605 and end section information 606 indicates, for example, respectively, which section of the test begins and ends the test.

[0165] Event information 607 indicates, for example, the order of events of the test for that form. Each event has a name and is prefixed with an event type and a colon. Other formats are optional. The event type includes "section", "report", and "results". Version information 608 and title information 609 indicate the version and title of the form, respectively. Skip allowed information 610 indicates, for example, whether or not by default skipping of sections is allowed. Restartable information 611 indicates, for example, whether the form can be restarted. Any optional, customized information regarding the form is stored in custom storage 616 as a property set or other data storage format. Timer storage 628 stores, for example, information relating to how the form is to be timed as a storage element. Attributes storage 630 stores, for example, the names of Timer Plugin 158 to be used with the form. Plugin data storage 632 and plugin data storage 633 store any data necessary for timer plugin 158 as a storage element and a stream of data, respectively. Plugin data storage 632 and plug in data storage 633 are optional. Scoring storage 634 stores, for example, information relating to the scoring of the form. Attributes storage 636 stores, for example, the name of scoring plugin 164 to be used with the form. Plugin data 638 and plugin data 639 optionally store any data needed for scoring Plugin 164 as a storage element and a stream of data respectively.

[0166] Items Branch 650, as seen in FIG. 14, stores, for example, the primary attributes, properties, and data that govern the items, or test questions, to be delivered to the examinee during the test. Items branch 650 can store information for various items, as is denoted by the three, vertical ellipses. A single item is identified by the data stored in name attributes storage 652. Again, the various items may each be identified by a different name, as denoted by the solid border around name attributes storage 652. Attributes storage 654 stores, for example, weight information 654, scored information 655, and optionally stores skip allowed information 656, title information 657, start information 658, finish information 659, and condition information 660. Weight information 654 indicates, for example, a value used for judging and scoring the item. In one embodiment, by default an item is given a weight of one in accordance with one embodiment, but other values may be utilized. Scored information 655 indicates, for example, whether or not the item is scored as opposed to whether the item is being used as an example. The default of scored information 655 is true. Skip allowed information 656 indicates, for example, whether the examinee can skip the item without answering.

[0167] Start information 658 indicates, for example, script execution at the beginning of the item and finish information 659 indicates, for example, script execution at the end of the item. Condition information 660 indicates, for example, whether or not there is a condition on the item being delivered to the examinee. The information stored in attributes storage 654 is stored as a stream of data or other data storage format. Data storage 662 and data stream 664 store any information regarding the properties of the item. For example, data storage 662 or data stream 664 can store the correct answer of a multiple choice item. Data storage 662 and data stream 664 stored the information as a storage element and a stream of data respectively.

[0168] Any optional, customized information regarding the item is stored in customs storage 666 as a stream of data or other data storage format. Category storage 668 stores, for example, information relating to each category to which the item belongs. The information stored in category storage 668 preferably and optionally is redundant, as category branch 700 stores, for example, all the items within the specific categories. The reason for the optional redundancy is so that test driver 110 can quickly look up the category of any item.

[0169] Category branch 700, as seen in FIG. 15, stores, for example, the primary attributes, properties, and data that govern the test categories. A test category provides a grouping mechanism, which is independent of delivery of the test, allowing for exotic reporting and scoring if necessary. Category branch 700 is optional as denoted by the broken border. Category branch 700 can store information for various categories, as is denoted by the three, vertical ellipses. A single category is identified by the data stored in name attributes storage 702. Again, the various categories may each be identified by a different name, as denoted by the solid border around name attributes storage 702. Attributes storage 704 stores, for example, complete information 705, duplicates information 706, contents information 707, and optionally stores, for example, description information 708. Complete information 705 indicates, for example, whether or not every item in the category must appear within the category or within its subcategories. Duplicates information 706 indicates, for example, whether the item can appear more than once within the category or within the subcategories. Contents information 707 determines what can exist within a category.

[0170] Description information 708 is used within the category to contain a description of the category's contents. Category storage 710 stores, for example, information relating to any subcategories under the category identified in name attribute storage 702. Items storage 712 indicates, for example, any items that exist within the category. Sections storage 714 contains information indicating what any sections that exist within the category. Scoring storage 716 contains information relating to the scoring of the items within the category. Attributes storage 718 stores, for example, the name of the scoring plugin to be used with the item. Data storage 720 and data stream 722 contain the information needed to initialize scoring plugin 164. Data storage 720 and data stream 722 store the information as a storage element and a stream of data respectively.

[0171] Templates branch 750, as seen in FIG. 16, stores, for example, the primary attributes, properties, and data that govern the templates used in the test. Template branch 750 can store information for various main templates, as is denoted by the three, vertical ellipses. A single main template is identified by the data stored in name attributes storage 752. Again, the various templates may each be identified by a different name, as denoted by the solid border around name attributes storage 752. Attributes storage 754 stores, for example, split information 756, order information 757, and optionally stores size information 759. Split information 656 defines how a specific area within the template is to be split or separated, for example, either by rows or columns or other shapes and/or sizes. Size information 759 indicates, for example, possible values for describing the size of the template, for example, pixels, percentages, or html syntax. Template storage 760 stores, for example, information relating to any sub-templates to be used under the templates specified by the information in name attributes storage 752. Sub-templates are identified by the information in name attributes storage 762. Many sub-templates 760 can exist as denoted by the three vertical ellipses.

[0172] Areas storage 764 indicates, for example, information relating to the areas used within the template denoted by the information in name attributes storage 752. Many areas may exist within a template as denoted by the three vertical ellipses. Each area is identified by the information stored in name attribute storage 766. Attribute storage 768 stores, for example, visible plugin name information 760, size information 770, and allow more information 771. Plugin name information 760 indicates, for example, the name of the visible plugin to be used with the area. Size information 770 indicates, for example, the size of the area, as for example a pixel value, a percentage value, or HTML syntax. Plugin data 772 and plugin data 774 store information relating to the visible plugin to be used in the area. The data stored in either plugin data storage 772 or plugin data stream 774 is executed by the visible plugin when the template is loaded. Plugin data storage 772 and plugin data stream 774 stores, for example, the information as either a storage element or a stream of data, respectively. Other information may optionally be stored.

[0173] Section branch 800, as seen in FIG. 17, stores, for example, the primary attributes, properties, and data that govern test sections. Test sections dictate the navigation and timing of groups of items as well as displays within the test. Sections branch 800 can store information for various sections, as is denoted by the three, vertical ellipses. A single section is identified by the data stored in name attribute storage 802. Again, the various sections may each be identified by a different name, as noted by the solid border around name attributes storage 802. Attributes storage 804 stores, for example, group information 805 and optionally stores title information 806, skip allowed information 807, start information 808, finish information 809, and condition information 810. Group information 805 indicates, for example, to which group of the test the section belongs. Skip allowed information 807 indicates, for example, whether or not the items within the section may be skipped. Start information 808 indicates, for example, script execution at the beginning of the section and finish information 809 indicates, for example, script execution at the end of the section. Condition information 810 indicates, for example, any conditions that exist regarding the section. Any optional, customized information regarding this section is stored in custom property storage 812 as a stream of data or other data storage format. Custom attributes will be stored as a property set. The "key" for each attribute will be a string or other acceptable format.

[0174] Timer storage 814 stores information regarding, for example, the timing of the section. Attribute storage 816 stores, for example, information identifying timer plugin 158, which is to be used with a section. Plugin data storage 818 and plugin data storage 820 stores, for example, data needed for timer plugin 158. Plugin data storage 818 and plugin data storage 820 stores, for example, information as a storage element and a string of data, or other acceptable format, respectively. Navigation storage 822 stores, for example, information relating to the delivery of presentations and groups within the section. Attributes storage 824 stores, for example, information indicating which navigation plugin 162 is to be used with this section. Plugin data storage 826 and plugin data stream 828 store information needed for the navigation plugin 162. Plugin data storage 826 and plugin data stream 828 store the information as a storage element and a stream of data respectively. Groups branch 850, as seen in FIG. 18, stores, for example, the primary attributes, properties, and data that govern the groups within the test. A group determines the order of events within the test. Groups branch 850 can store information for various groups, as is denoted by the three, vertical ellipses. A single group is identified by the data store in name attributes storage 852. The various groups may each be identified by a different name, as noted by the solid border around name attributes storage 852. Attributes storage 854 stores, for example, type information 855, event information 856, title information 857, and reviewed name information 858. Type information 855 indicates, for example, whether the group is either a "group holder" (group of presentations), or a "section holder" (group of sub-sections). These are mutually exclusive.

[0175] Event information 856 indicates, for example, the order of events within the test. Review name information 858 indicates, for example, whether or not a presentation within the group is to be used as a review screen. Any optional, customized information regarding the group is stored in custom storage 860 as a stream of data or other data storage format. Events storage 862 stores event information, for example, as is described in further detail in FIG. 19. Scoring storage 864 stores, for example, information relating to the scoring of items within the group. Attributes storage 866 stores, for example, information indicating which scoring plugin 164 is to be used with the group. Selection storage 872 stores, for example, information relating to the selection of items within the group. Attributes storage 874 indicates, for example, which selection plugin 160 is to be used with the group.

[0176] FIGS. 19A, 19B, 19C, and 19D illustrate the events sub-branch of groups branch 850 in greater detail, in accordance with one embodiment of the invention. In FIG. 19A, events sub-branch 862 can store information for various events. For example, events sub-branch 862 is storing information in events name sub-branch 880, event name sub-branch 890, and event name sub-branch 897. Attributes storage 881, in FIG. 19B, under events name storage 880 stores, for example, type information 882, template information 883, and optionally stores title information 884, counted information 885, start information 886, finish information 887, and condition information 888. Type information 882 indicates, for example, whether the event is an item or a display. Template information 883 indicates, for example, which template is being used with the event. Counted information 885 indicates, for example, whether a presentation should be included in the totals of presentations presented to the examinee in a section.

Generally, presentations with items, or questions, are counted and introductory presentations are not counted.

[0177] Start information 886, finish information 887, and condition information 888 indicates, for example, start, finish, and conditional scripts respectively. Any optional, customized information regarding the event is stored in custom storage 889. The "key" for each custom attribute will be a string. Referring again to FIG. 19A, event name storage 890 indicates, for example, a different event, which contains different attributes. Additionally, area information 891, in FIG. 19B, indicates, for example, which area is rendering the presentations content and item information 892 indicates, for example, the name of the associated item if the event is of the item type. Additionally, data storage 893, data stream 894, data storage 895, and data storage 896 contain information used in a nested presentation. The data off of a nested presentation are the contents of the item or the presentation. This data may be a stream, a storage, a link to a stream, a link to a storage, or other format. In FIG. 19C, event name 897 indicates, for example, another event, which includes a sub-event 898, in FIG. 19D.

[0178] Plugins branch 900, as seen in FIG. 20, stores, for example, the primary attributes, properties, and data that govern any plugins 150 used for the test. Plugins branch 900 can store information for various plugins, as is denoted by the three, vertical ellipses. A single plugin is identified by the data stored in name attribute storage 902. A CLSID is stamped with the name of the plugin 150. Attributes storage 904 stores, for example, information identifying the plugin 150 by a program ID. Data storage 906 and data storage 908 store initial data for the plugin as either a storage element or a stream of data respectively.

[0179] Data branch 950, as indicated in FIG. 21, stores, for example, any global data needed for the test. Data stored optionally under data branch 950 may be stored as either a storage element or a stream of data as indicated by data storage 952 and data storage 954. Data stored under data branch 950 may be directly used by a plugin 150 or the data may be resources (.gif, .jpeg, .wab, .mpeg, etc.) used internally by a plugin 150.

[0180] FormGroups branch 1000, as seen in FIG. 22, stores, for example, the primary attributes properties and data that govern the formGroups of the test. FormGroups branch 1000 can store information for various formGroups, as is denoted by the three, vertical ellipses. A single formGroup is identified by the data stored in name attributes storage 1002. The various formGroups may each be identified by a different name, as denoted by the solid border around name attributes storage 1002. Attributes storage 1004 stores, for example, information indicating which forms are to be used within the formGroup. Selections storage 1006 stores, for example, information relating to the selection of items within the formGroup. Attributes storage 1008 indicates, for example, which selection plugin 160 is to be used with the formGroup. Plugin data storage 1010 and plugin data storage 1012 store any information needed for the selection plugin 160. Attributes storage branch 1050 stores, for example, attribute information that is global to exam resource file 120. This includes the last execution state of XXL compiler 140 [sMode], the major [iXXLMajorVersion] and the minor version [iXXLMinorVersion] of the XXL language.

[0181] Scripts branch 1100 stores, for example, information relating to scripts used within the test. Attributes storage 1102 stores, for example, type information that specifies which type of language the script is in, for example, VB script of J script. Scripts storage 1104 stores, for example, global scripts used within the test that may be referenced by the test driver. MsgBox branch 1150 stores, for example, information relating to the size and content of any message boxes that may be delivered to the examinee during the test. Message boxes may be triggered by plugins 150 during the exam.

[0182] D. POLESS Exam Instance File

[0183] FIGS. 26A, 26B, 26C, and 26D illustrate the POLESS layout of exam instance file 170. Exam instance file 170 stores information regarding the current examinee's test. Exam instance file 170 is created when a test starts for an examinee. Exam instance file 170 is destroyed when the test successfully completes. If the examinee must restart her test due to some interruption, for example, a power failure, the state of the test is restored from Exam instance file 170. In a preferred embodiment, the layout of exam instance file 170 is in a hierarchical POLESS format. As seen in FIG. 26A, the top-level storage branches of exam instance file 170 from root 1200 are, for example: running branch 1202; contents branch 1310; and history branch 1320. Root 1200 relates to POLESS cStorageRoot class 406 (FIG. 10), which instantiates exam instance file 170.

[0184] Running branch 1202 stores, for example, the state information of all running objects in test driver 110 and plugins 150. Plugins 150 use one of IPersistInstanceStream interface 196a, IPersistInstanceSet interface 196b, or IPersistInstanceStore interface 196c to store information to exam instance file 170 as a stream of data, a set of data, or a store of data, respectively. Any of plugins 150, except display plugin 152, results plugin 166, report plugin 168, and helm plugin 154, which do not contain examination state information, store examination state information to exam instance file 170. Test driver 110 determines the storage location in exam instance file 170 that stores a particular piece of examination state information.

[0185] Exam sub-branch 1204 contains examination state information relating to the exam. Contents storage 1206 stores, for example, exam status information 1207 and version information 1208. Exam status information 1207 indicates, for example, the status of the exam, for example, initializing or terminating. Template storage branch 1210 stores, for example, examination state information relating to templates running in the exam. Name attribute storage 1212 stores, for example, count information 1214 and observed ever information 1215. Observed ever information 1215 indicates, for example, whether or not the template's content has ever been fully seen by the examinee.

[0186] Form storage branch 1216 contains information relating to the forms used within the exam. Contents storage branch 1218 stores, for example, seconds information 1219, date start information 1220, date finish information 1221, current section information 1222, and version information 1223. Current section information 1222 indicates, for example, the current section being delivered to the examinee in the form. Version information 1223 indicates, for example, the identification of the form.

[0187] Sections chosen storage branch 1224, as illustrated in FIG. 26B, stores, for example, information relating to sections in the form being delivered to the examinee. Contents storage 1226 stores, for example, the names of the sections that have been or will be delivered to the examinee. Name attribute storage 1228 indicates, for example, the name of a particular section. Contents storage 1230 stores, for example, current child information 1231, seconds information 1232, date start information 1233, and date finish information 1234. Navigation storage 1236 and navigation storage 1237 store the state information of navigation plugin 162. Navigation storage 1236 stores, for example, the examination state information from navigation plugin 162 if navigation plugin 162 implements the IPersistInterfaceSet 196b or IPersistInterfaceStore 196c. Navigation storage 1237 stores, for example, the information from navigation plugin 162 if navigation plugin 162 implements IPersistInterfaceStream 196a. Timers storage 1238 and timers storage 1239 store information from timer plugin 158. Timer storage 1238 is used if timer plugin 158 implements IPersistInterfaceSet 196b or IPersistInterfaceStore 196c. Timers storage 1239 is used if timer plugin 158 uses IPersistInterfaceStream 196a.

[0188] Items chosen sub-branch storage 1240 stores, for example, information relating to items that have been or will be delivered to the examinee. Contents storage branch 1242 stores, for example, the names and order of all the items that have been or will be delivered to the examinee. Name attributes storage 1244 indicates, for example, the identification of a particular item. Contents storage branch 1246 stores, for example, presented information 1244, complete information 1248, skipped information 1249, seconds information 1250, dehydrated information 1251, and observed ever information 1252. Presented information 1247 indicates, for example, whether the item has ever been delivered to the examinee. Completed information 1248 indicates, for example, whether or not the item has been completed. Skipped information 1249 indicates, for example, whether the item has been skipped. Item plugin storage 1254 and item plugin storage 1255 stores, for example, examination state information from item plugin 156. Item plugin storage 1254 is used if item plugin 156 uses IPersistInterfaceSet 196b or IPersistInterfaceStore 196c. Item plugin storage 1255 is used if item plugin 156 uses IPersistInterfaceStream 196a.

[0189] In FIG. 26C, item light storage 1256 exists only if the item was dehydrated (to save memory or when a section ends). The dehydrated item stores the data but actions on the data are no longer available until the item is re-hydrated. Item light storage 1256 stores, for example, score candidate information 1257. Score minimum information 1258, score nominal information 1259, score maximum information 1260, complete information 1261, skipped information 1262, correct answer display 1263, response results 1264, and correct answer results 1266. Timers storage 1268 and timers storage 1269 store information from timer plugin 158. Timer storage 1268, as seen in FIG. 26B, is used if timer plugin

158 implements IPersistInterfaceSet 196b or IPersistInterfaceStore 196c. Timers storage 1269 is used if timer plugin 158 uses IPersistInterfaceStream 196a. Score storage 1270 and Score storage 1271 store information from timer plugin 158. Timer storage 1270 is used if timer plugin 158 implements IPersistInterfaceSet 196b or IPersistInterfaceStore 196c. Score storage 1271 is used if timer plugin 158 uses IPersistInterfaceStream 196a.

[0190] In FIG. 26C, groups chosen sub-branch storage 1272 indicates, for example, which groups have been or will be delivered to the examinee. Contents storage 1274 stores, for example, the names of the groups. Name attributes storage 1276 indicates, for example, the name of a particular group. Contents storage 1278 stores, for example, names of groups and the order of groups. Scoring storage 1280 and scoring storage 1281 store examination state information from score plugin 164. Scoring storage 1280 is used if score plugin 164 implements IPersistInterfaceSet 196b or IPersistInterfaceStore 196c. Scoring storage information 1281 is used if score plugin 164 implements IPersistInterfaceStream 196a. Selection storage 1282 and selection storage 1283 store information from selection plugin 160. Selection storage 1282 is used if selection plugin 160 implements IPersistInterfaceSet 196b or IPersistInterfaceStore 196c. Selection storage 1283 is used if selection plugin 160 implements IPersistInterfaceStream 196a. Delivered storage 1284, in FIG. 26D, stores, for example, an ordered list of groups chosen for delivery. Delivered storage 1285 stores, for example, an ordered list of the sub-classes of the form, for example: sections, reports and results.

[0191] Presentations chosen storage sub-branch 1286 indicates, for example, any presentations that have been or will be delivered to the examinee. Contents storage 1288 stores, for example, the names of the presentations. Names storage sub-branch 1290 stores, for example, the name of the presentation. Names storage 1290 also stores, for example, comment information 1291, marked information 1292, count information 1293, name information 1294, observed ever information 1295, name information 1296, and observed ever information 1297. Name information 1294 and observed information 1295 relate to the name of the first presentation area stored under presentations chosen sub-branch 1286 and whether or not the presentation has ever been observed, and name information 1296 indicates, for example, the last presentation area that was delivered to the examinee and whether or not the presentation was ever observed. Contents storage 1298 stores, for example, information leading to events. Contents storage 1298 stores, for example, ready information 1299 ever checked information 1300, ever started information 1301, and ever finished information 1302. Ready information 1299 indicates, for example, whether the event is ready to be delivered to the examinee. Ever checked information 1300 indicates, for example, whether an event's conditional delivery script ever been checked. Preferably, the conditional delivery script is only checked once. Ever started information 1301 indicates, for example, whether the event was ever started by the examinee. Ever finished information 1302 indicates, for example, whether the event was completed by the examinee.

[0192] Referring again to FIG. 26A, contents branch 1310 stores, for example, a property set containing information to identify the examination instance and the examination start count 1312. The identifying information used is the examinee appointment identification 1311, the name 1313 of exam resource file 120, and the name 1314 of the specified form or group.

[0193] History branch 1320 is a single stream of chronological text messages that logs the history of the test. These text messages are used by staff at system headquarters to diagnose problems that occurred in the field. Each text message is prefixed with the date, time, and a level of severity, for example: information, warning, or error. Test driver 110 will filter the text messages to a level of diagnostics desired for test driver 110, such as determining errors in test driver 110 or detail history tracking, including general information.

[0194] V. Expansion of Test Driver Using Plugins

[0195] FIG. 27 illustrates the process for customizing test based on specific requirement from the client using plugins 150, denoted generally by reference numeral 1400. First, the client presents the new requirements, for example, a new item type, to the test developer, step 1402. The test developer then writes and XML schema to define the XXL test specification, step 1404. The schema is subsequently used to validate the XXL test specification.

[0196] A detailed description of the XXL schema is given in U.S. Patent Publication No. 20030129573, entitled

"EXTENSIBLE EXAM LANGUAGE (XXL) PROTOCOL FOR COMPUTER BASED TESTING," incorporated herein by reference.

[0197] The test developer next writes the appropriate plugin 150, in this example, item plugin 156. The test developer also implements the IPlugin interface 167 and IPlugin interface and IItem interfaces 169. Additionally, the test developer implements IPersistResource interface 192 (FIG. 3) to enable persistence of compiled test information from item plugin 156 to exam resource file 120. The test developer can optionally implement IPersistInstance interface 196 (FIG. 3), step 1408, to enable persistence of examination state information from item plugin 156 to exam instance file 170. After the appropriate interfaces have been implemented, item plugin 156 is valid and operating. Finally, after the test is delivered to the examinee, the result processor accumulates results from the examinee, 1410. The results processor must be able to understand the new item type to correctly process the results. Customization process 1400 only required the test developer to write one piece of software, item plugin 156, to accommodate the client's customizations rather than multiple pieces of software.

[0198] A. Test Production and Test Delivery

[0199] FIG. 28 is a flow chart illustrating the overall method of test production and test delivery, denoted generally by reference numeral 1500. The test publisher first authors the test specification and content, step 1502. The test specification and content are then stored in exam source files 130, step 1504. Exam source files 130, for example the content of XXL files 134, are then compiled and validated, step 1506. The compiled test specification and content are stored in exam resource file 120, step 1508. Finally, the compiled test specification and content are delivered to the examinee, step 1510.

[0200] The validation of the test specification and content is illustrated in greater detail in FIG. 29, by the method denoted generally by reference numeral 1512. When the test specification and content stored in exam source files 130 specifically references a plugin 150, that plugin 150 is instantiated, step 1514. Partial test specification and content relating to that plugin 150 are loaded into the plugin 150 from exam source files 130, step 1516. In an alternative embodiment, the partial test specification and content are loaded into a private memory in data communication with the plugin 150. The plugin 150 validates the partial test specification and content, step 1518. The validated test specification and content are then unloaded from the plugin 150 into a storage element within exam resource file 120.

[0201] FIG. 30 illustrates the method of the test delivery cycle in greater detail. When the previously validated test specification and content stored in exam resource file 120 references a plugin 150, the plugin 150 is instantiated, step 1525. The storage element in exam resource file 120 containing the validated test specification and content are provided to the plugin 150, step 1527. The validated test specification and content are loaded into the plugin 150 from the storage element within exam resource file 120, step 1529. Finally, the examination state information, which includes, for example, the examinee's responses, is stored into exam instance file 170, step 1533.

[0202] FIG. 31 illustrates the method of restarting a test after interruption in greater detail. In test restart method 1535, test driver 110 is started, step 1537. Test driver 110 determines whether the test has already started, step 1539. If the test delivery has not already started, plugins 150 reload validated test specification and content from exam resource file 120, step 1543. If the test has already started, plugins retrieve examination information from exam instance file 120, step 1541. Plugins 150 then reload the validated test specification and content from exam resource file 120, step 1543. Test driver 110 then delivers the exam to the examinee, step 1545.

[0203] B. Plugin Life Cycle

[0204] FIG. 32 illustrates the life cycle of plugin 150 from test production to test delivery, denoted generally by reference numeral 1420. Dashed vertical line 1422 divides the plugin life cycle 1420 into a test production cycle, to the left of dashed vertical line 1422, and a test delivery cycle, to the right of dashed vertical line 1422. The test production cycle occurs only occasionally when new plugins 150 are developed to satisfy the requirements of a client. The test delivery cycle occurs whenever the test is delivered to the examinee, for example, daily.

[0205] Exam source files 130, of which data files 132 and XXL files 134 are shown, contain every aspect of the test as written by the test publisher. In step I, XXL compiler 140 reads from XXL files 134 and interprets instructions that call for the use of a plugin 150. Plugin 150 is identified in the XXL test definition language by both a name and a program identification ("prog ID"). When XXL compiler receives the prog ID from XXL files 134, XXL compiler knows that a plugin 150 is required to complete the compilation of exam source files 130.

[0206] Not all of the possible types of plugins 150 are required to build any one test. Also, more than one plugin 150 is implemented for a specific type. In the above example, two navigation plugins 162 and two item plugins 156 are defined. XXL compiler 140 reads information from exam source files 130 using IStream interface 340, iNode interface 1424, which is the Microsoft interface used to access a node of an XML document in the document object model ("DOM"), and IStreamVB interface 348. XXL compiler 140 instantiates the requested plugin 150 using, for example, the call CoCreateInstance( ). CoCreateInstance( ) creates a single, uninitialized object of the class associated with a specified CLSID, using a prog ID that has been converted into the CLSID.

[0207] If the data referring to plugin 150 has been customized by the test developer, XXL compiler 140 may not recognize the new data. Therefore, XXL compiler 140 passes the data directly to plugin 150 and plugin 150 loads the data into a private memory (not shown). In one embodiment, the private memory is internal to plugin 150, and in another embodiment, the private memory is external to plugin 150. Plugin 150 can then validate the data using the XXL schema. If the data is invalid, plugin 150 reports the error. In an alternative embodiment, plugin 150 can validate the data using an XML document type definition ("DTD"). A DTD is a formal description in XML Declaration Syntax of a particular type of document. Similar to a schema, a DTD sets out what names are to be used to the different types of elements, where they may occur, and how they all fit together. However, the XXL schema is preferred for validation since schemas are easier to read than a DTD and are very flexible.

[0208] If plugin 150 declares that the data is valid, XXL compiler 140 prepares a POLESS storage object 300 in exam resource file 120 to which plugin 150 saves the data at a command from XXL compiler 140, in step II. As described previously, XXL compiler 140 determines where the data from plugin 150 is to be saved in exam resource file 120 and creates the appropriate storage location. The name, CLSID, and data associated with plugin 150 is stored in plugins branch 900 in exam resource file 120 (FIG. 20). Plugin 150 implements IPersistResource interface 192 to store the data to exam resource file 120. Data storage 906 stores, for example, the data, for example, as either a stream, set of data, or as a storage element if plugin 150 implements either IPersistResourceStream 192a, IPersistResourceSet interface 192b, or IPersistResourceStore interface 192c, respectively. Data storage 908 stores, for example, the data as a stream of data if plugin 150 implements IPersistResourceStream interface 192a. Plugin 150 can choose the format used to store the data into exam resource file 120. Steps I and II are repeated until exam source files 130 are completely compiled and exam resource file 120 is completely populated with the compiled test information.

[0209] The compile sequence of a plugin 150, as shown in steps I and II in FIG. 32, are illustrated in greater detail in FIG. 33. Plugin compile sequence 1430 begins as XXL compiler 140 asks plugin 150 to validate the information from exam source files 130 that pertain to plugin 150 using IPlugin::ValidateSource( ) call 1432, in step I. Plugin 150 validates whether or not the data received from exam source files 140 is correctly formatted based on the XXL schema. If the data is not valid, plugin throws a structured COM error.

Plugin 150 does not validate that all required source elements are present, but rather, that what is present is correctly formatted.

[0210] Step II contains two steps, indicated as step IIa and IIb. In step IIa, XXL compiler 140 creates the appropriate storage element in exam resource file 120 using POLESS object 300. The storage element type is determined based on the type of IPersistResource interface 192 that plugin 150 implements, for example: IPersistResourceStream interface 192a; IPersistResourceSet interface 192b; or IPersistResourceStore interface 192c. XXL compiler 140 then calls IPersistResource*::Save( ) call 1434 for the appropriate IPersistResource interface. Plugin 150 saves the compiled information from exam source files 130 to exam resource file 120 through the POLESS object 300 passed by XXL compiler 140. In step IIb, XXL compiler 140 instructs plugin 150 to unload, or flush, its content using Unload( ) call 1436. As stated previously, steps I, IIa, and IIb are repeated until all of exam source files 130 is compiled.

[0211] Step VI, which is shown as steps VIa and VIb, concerns amalgamation of exam resource file 120. Amalgamation enables data for a specific plugin to exist virtually as one storage location even if the data appears at different locations within the storage hierarchy. Amalgamation can be performed on exam resource file 120 if plugin 120 has implemented either IPersistResourceSet interface 192b or IPersistResourceStore interface 192c which storing data to exam resource file 120. In step VIa, XXL compiler 140 amalgamates one to three storage elements in exam resource file 120 and passes the amalgamated POLESS object to plugin 150 using IPersistResource*::ValidateResource( ) call 1438. Plugin 150 determines whether or not the amalgamated POLESS object creates a complete and valid set. Plugin 150 throws a structured COM error if the amalgamated POLESS object does not create a complete and valid set. In step VIb, XXL compiler 140 instructs plugin 150 to unload, or flush, its content using Unload( ) call 1440. Steps VIa and VIb are interspersed among steps I, IIa, and IIb cycles and can also occur multiple times during the compilation of exam source files 130. Amalgamation is described in greater detail, in U.S. Patent Publication No. 20030129573, entitled "EXTENSIBLE EXAM LANGUAGE (XXL) PROTOCOL FOR COMPUTER BASED TESTING," incorporated herein by reference.

[0212] Referring again to FIG. 32, during the test delivery cycle, test driver 110 reads the test specifications stored in exam resource file 120 through POLESS objects 300. Test driver 110 reads information from exam resource file 120 through POLESS objects 300 in order to retrieve the encrypted, compressed, and structured elements within exam resource file 120. When the XXL test definition language calls a plugin 150 by a prog ID, as described previously, test driver 110 instantiates the plugin 150 that was called, in step III. Test driver 110 provides the POLESS object 300 from exam resource file 120 and plugin 150 initializes itself from the POLESS object 300, for example, data storage 906 or data storage 908 stored under name attribute storage 902, using the appropriate IPersistResource interface 192. The information loaded into plugin 150 is the same information as was stored into exam resource file 120 by plugin 150 during the test production cycle (step II). Since plugin 150 chose the storage format used to store the information into exam resource file 150, plugin 150 can always read the information from exam resource file 150, giving plugin 150 complete flexibility. Test driver 110 need not be able to read the information that is used by plugin 150. Therefore, any customizations to the test facilitated by plugin 150 does not require any changes to test driver

110. The test then progresses with plugin 150 enhancing the functionality of test driver 110 based on the new requirements from the client.

[0213] Periodically, based on a request either from test driver 110 or from plugin 150, the state of all running objects will save to exam instance file 170, which is a unique file for each examinee, indicating the progress and the status of the test for that examinee. Test driver 110 asks plugin 150 if plugin 150 is "dirty," meaning that plugin 150 is storing has some updated examination state information. For example, when the examinee selects distractor A on a multi-choice item, item plugin 156, in this case, becomes dirty. If plugin 150 is dirty, test driver 110 provides plugin 150 a POLESS object 300 in exam instance file 170 and plugin saves the examination state information to exam instance file 170 using IPersistInstance interface 196, in step IV. For example, item plugin 156 saves the examinee's answer to item plugin storage 1254 or to item plugin storage 1255 (FIG. 26). Item storage 1254 stores, for example, the data as either a set of data or as a storage element if item plugin 156 implements either IPersistInstanceSet interface 196b or IPersistInstanceStore interface 196c, respectively. Item storage 1255 stores, for example, the data as a stream of data if item plugin 156 implements IPersistInstanceStream interface 196a.

[0214] Step V occurs if the test is interrupted, for example, because of a power failure, and the test needs to restart. When test driver 110 is required to return to a particular operation state, test driver 110 reads the examination state information from exam instance file 170. Plugin 150 is provided the storage object containing the state of plugin 150 as saved in step IV using IPersistInstance interface 196. Using the previous example, item plugin 156 retrieves its state information from item plugin storage 1254 or for item plugin storage 1255. Plugin 150 is able to become operational from the retrieved state information, enabling a restart of the test from the point at which the test was interrupted.

[0215] The delivery sequence of a plugin 150, as shown in steps II, IV, and V in FIG. 32, are illustrated in greater detail in FIGS. 34A, 34B, 34C, and 34D. As seen in FIG. 34A, delivery sequence 1520 particularly relates to visible plugins 150, e.g., display plugin 152, helm plugin 154, and item plugin 156. Step III contains sub-steps labeled IIIa-IIIb. Plugin delivery sequence 1520 begins, in step IIIa, when the current delivering presentation requests its template to activate with cTemplate::Activate( ) call 1524. Activate( ) call 1524 is activated when the examinee navigates on the test using a helm navigation control activated by helm plugin 154. IContainerNotifyHelm interface 206 allows helm plugin 154 to request navigation from test driver 110. IContainerNotifyHelm interface 206 sends Activate( ) call 1524 to cTemplate class 236 in test driver 110 (see FIG. 8).

[0216] In step IIIb, cTemplate class 236 in test driver 110 uses IPlugin::Load( ) call 1526 to set the core object references from test driver 110 into the plugin 150 being delivered. The core object references include IContainerNotify interface 200, the cExam class (not shown), and the IAppointment interface 176, which passes information regarding the examinee and appointment to plugin 150.

[0217] Step V, which is interspersed with step III, occurs only if the test is interrupted and plugin 150 loses state. cTemplate class 236 in test driver 110 uses IPersistInstance*::Reload( ) call 1528 to call on the reload method of exam instance file 170. Exam instance file 170 reloads plugin 150, through IPersistInstance interface 192, for example, IPersistInstanceSet 192b, with the state saved to the appropriate storage location in exam resource file 170 (see FIG. 26).

[0218] Step IIIc is performed for both initial delivery of plugin 150 and during restart of the test, in conjunction with step V. cTemplate class 236 in test driver 110 uses IPersistResource*::Load( ) call 1530 to call on the load method of exam resource file 120. Exam resource file 120 loads plugin 150, through IPersistResource interface 192, for example IPersistResourceSet interface 192b, with the test specification and content from the appropriate storage location in exam resource file 120. Plugin 150 is loaded with test specification and content from exam resource file 120 when being initially delivered to the examinee. Plugin 150 is also loaded with test specification and content from exam resource file 120 and with examination state information from exam instance file 170, as described above, when the test has been interrupted and plugin 150 must recover state.

[0219] After plugin 150 is properly loaded, cTemplate class 236 in test driver 110 uses, I*::PresentationStarting( ) call 1532 (continued in FIG. 34B) to inform visible plugin

150 that the presentation is starting, in step IIId. I*::PresentationStarting( ) call 1532 is made to any visible plugin 150 being used in the presentation on the appropriate interface, for example: IDisplay interface 169a, IItem interface 169c, or IHelm interface 169b. For example, an IItem::PresentationStarting( ) call is used for item plugin 156. cTemplate class 236 then instruct visible plugins 150 to display using IOleObject::DoVerb(Show, . . .) command 1534, step IIIe. IOleObject interface 1522 is the Active Document interface used to implement the Active Document presentation. IOleObject interface 1522 is the combination of the Active Document interfaces described in conjunction with FIG. 7. After instructing visible plugins 150 to display, test driver 110 awaits notification from each visible plugin 150 that the specific visible plugin 150 has successfully shown. Visible plugins 150 call back to test driver 110 using IContainerNotify::Activated( ) call 1536, step IIIf (continued in FIG. 34B). Now, the presentation is started and active such that the examinee can interact with the presentation.

[0220] The deactivation of the presentation begins with a request from the helm for navigation. For example, if the examinee has finished a question and wishes to move on to the next question on the next presentation, the examinee can choose the "NEXT" button on the helm. The navigation request is sent from IHelm interface 169b, which receives the request from the examinee, to test driver 110 using IContainerNotifyHelm interface 206. As seen in FIG. 34D, the request is made using IContainerNotifyHelm::Request- Move( ) call 1538, step IIIg. Test driver 110 then asks each item plugin 156 being used in the presentation template if the examinee is allowed to leave the current presentation and to proceed to the next presentation. The query is made using IItem::bProceed( ) call 1540, step IIIh. If all item plugins 156 respond in the affirmative, test driver 150 passes the navigation request to navigation plugin 162, which is an invisible plugin 150. Test driver 110 passes the request using INavigate::RequestMove( ) call 1542, step IIIl. Navigation plugin 162 determines the resultant location of the requested navigation. In FIG. 34, for example, navigation plugin 162 determines the section of the test to which the examinee will proceed using ISection::ChildNext( ) call 1544, step IIIj.

[0221] The active presentation then instructs the template to deactivate using cTemplate::Deactivate( ) call 1546, step IIIk (continued in FIG. 34C). Referring back to FIG. 34D, template class 236 in test driver 110 requests that visible plugins 150 hide from the Active Document using IOleObject::DoVerb(Hide, . . .) call 1548, step IIIl. cTemplate class 236 in test driver 110 informs visible plugins 150 that the current presentation is ending using I*::PresentationEnding( ) call 1550, step IIIm. For example, cTemplate informs helm plugin 154 that the current presentation is ending using the IHelm::PresentationEnding( ) call.

[0222] Step IV, which contains sub-steps IVa-c, is the process to save plugin state data to exam instance file 170. Test driver 110 requests the "dirty" state of plugin 150 to determine whether plugin 150 is storing any state information that would be necessary if the test were to be interrupted. Test driver 110 uses IPersistInstance*::IsDirty( ) call 1552 to make the request, step IVa. For example, test driver 110 uses IPersistInstanceSet::IsDirty call 1552 if the state data is a property set. If plugin 150 is storing state data that is not already stored in exam instance file 170, IPersistInstance*::IsDirty( ) call 1552 returns true. If plugin 150 is "dirty", test driver 110 instructs plugin 150 to save the state data to exam instance file 170 in the

POLESS object provided (FIG. 26) using IPersistInstance*::Save( ) call 1554, step IVb. Finally, test driver 110 instructs plugins 150 to unload all objects using IPlugin::Unload( ) call 1556, step IVc.

[0223] VI. Network Environment for a Computer-Based Testing System

[0224] A description of a network environment for a computer-based testing system according to the present invention including a test driver that controls delivery of a computer-based test over a networked environment by caching test components for delivery to a test candidate in order to facilitate a uniform testing environment for at least one or more concurrent test candidates is provided. FIG. 35 is a block diagram illustrating an example of a network environment for a computer-based test system according to the present invention.

[0225] With reference to FIG. 35, the computer-based testing system of the present invention includes a test management system, including a Web certified proctor management site 3160 and secure ID server 3150, for identifying proctors of the computer-based test; a test proctor certification system, including Web eligibility management site 3190 and GEE server 3180, for certifying proctors for administering the computer-based test; a test registration system to enable candidates to request an appointment to take a test, including Web registration site 3200 to allow appointments to be reserved via the Internet, scheduling middleware 3170 to process the candidates request to take the test, schedule interface service 3220 which provides the scheduled appointment data to candidate workstation 3000 to identify the candidate and to verify the candidate's credentials, an ecommerce system 3210 for registering test candidates for the computer-based test and for collecting and/or processing any payments associated with scheduling and delivering the computer-based test; and a test analysis system, including results collection server 3230 for aggregating results from all delivered tests, a data warehouse 3240 for storing and allowing access to such test results, and a results psychometric system 3250 for monitoring candidate responses to test items, maintaining the composition of the computer-based tests, and to enable item response analysis and reports to be generated and distributed.

[0226] The network environment as shown in FIG. 35 further includes a computer-based test production system, including test publisher system 3010 for authoring a computer-based test and producing exam source files (previously described) which store test specification and content; test packaging system 3020 for retrieving, validating and compiling the exam source files of the computer-based test into an exam resource file (previously described); and source server 3030 for storing and managing versions of the compiled exam resource file and any additional corresponding test components comprising the computer-based test. The network environment also includes distribution servers for facilitating computer-based testing, including application deployment servers 3040, item cache servers 3050 and plugin cache servers 3060, connected to source server 3030 via a wide area network (i.e., an interconnected system of networks), and connected to candidate workstation 3000 via a local access network. The local access network can include LAN, xDSL access network, cable access network, or wireless access network. From source server 3030, the test components are extracted from the exam resource file of the computer-based test and deployed to the distribution servers. Application deployment servers 3040 store the test driver application and the exam resource file, containing the core test specifications of the computer-based test, for distribution to candidate workstation 3000. Item cache servers 3050 store test items extracted from the exam resource file. Plugin cache servers 3060 store plugins extracted from the exam resource file. The network environment further includes any number of test candidate workstations at a multiplicity of locations to which the computer-based test is scheduled to be delivered. Thus, the distribution servers may be clustered for parallel processing, load balancing and fault tolerance to support a varied volume of at least one or more concurrent test candidates.

[0227] From application deployment servers 3040. the test driver application for controlling delivery of the computer-based test may be downloaded for setup on candidate workstation 3000. The test driver application 3070 may be stored on a computer readable medium, such as a hard drive or other magnetic medium, connected to candidate workstation 3030. However, test components of the computer-based test, including the exam resource file, test items and plugins (all of which were previously described), are stored in random access memory (RAM) to prevent unauthorized copying or manipulation of the computer-based test, thereby assuring the integrity of the test.

[0228] Test driver application 3070 includes session manager 3080 for managing the computer-based testing session, having authentication layer 3090 for authenticating proctors of the computer-based test, event locator candidate credential interface 3100 for certifying test candidate eligibility to use the computer-based test and Unified Test Driver (UTD) core 3110 for controlling delivery of the computer-based test to candidate workstation 3000. UTD core 3100 controls item cache controller 3120, plugin cache controller 3130 and browser presentation layer 3140. Item cache controller 3120 stores test items retrieved from item cache servers 3040 in item cache 4360 (FIG. 36), encrypted memory mapped paged RAM on candidate workstation 3000, for delivery to the test candidate. Plugin cache controller 3130 stores plugins retrieved from plugin cache servers 3160 in plugin cache 4390 (FIG. 36), memory mapped paged RAM on candidate workstation 3000, for use by UTD core 3100. Browser presentation layer 3140 serves the computer-based test on candidate workstation 3000 in accordance to the exam resource file retrieved from application deployment servers 3040.

[0229] VII. Caching Architecture for the Computer-Based Testing System

[0230] In computer-based testing, a test may be delivered to a candidate workstation using two delivery modes: a disconnected mode (e.g., non-networked) or a connected mode (e.g., networked). A disconnected mode is a traditional delivery mode in which every element required to administer the computer-based test is stored on the test candidate workstation before the computer-based test is initiated. Thus, there is no need to monitor the testing environment to adjust delivery of the computer-based test over a network environment and no need to implement a caching architecture for a network environment. In contrast, a connected mode is a non-traditional delivery mode in which core elements of the computer-based test are stored on the test candidate workstation and additional test components are retrieved during computer-based testing from distribution servers over a network environment. While using the connected mode provides many advantages, such as, flexibility in administering computer-based tests to test candidates located in multiple locations, the network environment introduces environment variables that are not controllable by a test administrator without great cost. Thus, in order to provide a uniform testing environment to a varied volume of at least one or more concurrent test candidates located in multiple locations, a computer-based system must adjust delivery of a computer-based test in order to compensate for variance in the network environment.

[0231] A description of a caching architecture for a computer-based testing system of the present invention including a test driver that controls delivery of a computer-based test over a networked environment by caching test components for delivery to a test candidate in order to facilitate a uniform testing environment for at least one or more concurrent test candidates is provided. FIG. 36 is a block diagram of a caching architecture for a computer-based test system according to the present invention.

[0232] As shown in FIG. 36, test driver application 3070 includes Web service interface 4400, having authentication module 4410. Web service interface 4400 facilitates network communications between candidate workstation 3000 and the distribution servers. Authentication module 4410 authenticates the test components retrieved from the distribution servers. Web service interface 4400 further logs the transmission details of each request and response transmitted between candidate workstation 3000 and the distribution servers, which includes, but is not limited to, for example, the time at which candidate workstation 3000 sends a request to a distribution server, the time at which a distribution server receives the request, the time at which a distribution server sends a response to candidate workstation 3000 and the time at which candidate workstation 3000 receives the response. Such details further include, for example, error messages issued by the network, error messages issued by the distribution servers, the number of retries by candidate workstation 3000 to send a request to the distribution servers, the number of retries by candidate workstation 3000 to obtain a response from the distribution servers and the size of each message or test components transmitted.

[0233] Test driver application 3070 further includes item request interface 4340, plugin request interface 4370, request processor 4310, decryption module 4320, decompression module 4330, item request module 4350, plugin request module 4380, cache controller 4400, item cache 4360 and plugin cache 4390. Test driver application 3070 sends requests to retrieve test components via item request interface 4340 and plugin request interface 4370. Request processor 4310 processes requests for test components initiated by test driver application 3070 and delivery of test components retrieved from the distribution servers to test driver application 3070. Item request module 4360 facilitates retrieval of test items from item cache servers 3040. Plugin request module 4380 facilitates retrieval of plugins from the plugin cache servers 3060. Cache controller 4400 manages the storing of test components in item cache 4360 and plugin cache 4390. Item cache 4350 stores test items retrieved from item cache servers 3040. Plugin cache 4390 stores test plugins retrieved from plugin cache servers 3060. Decryption module 4320 decrypts test components that have been encrypted for preserving the integrity of the computer-based test. Decompression module 4330 decompresses test components that have been compressed for transmitting over the network environment.

[0234] Descriptions of example caching operations of a computer-based testing system of the present invention including a test driver that controls delivery of a computer-based test over a networked environment having the caching architecture are provided. Because the computer-based test comprises cacheable objects, it is possible to download only selected test components from the distribution servers to the candidate workstation for delivering a current test section in accordance to the test specifications. Thus, it is possible for a test candidate to initiate a computer-based test prior to all test components being downloaded to the candidate workstation.

[0235] FIG. 37A is a flow chart of a method of caching test items with respect to the caching architecture according to the present invention. With reference to FIG. 37A, when test driver application 3070 requires a test item, test driver application 3070 via item request interface 4340 instructs request processor 4310 to retrieve the test item at steps S3710A and S3715A. Request processor 4310 verifies whether the test item is available in item cache 4360 at step S3720A. If the requested test item is available locally (e.g., hit), request processor 4310 retrieves the test item from item cache 4360 at step S3755A. If necessary, decompression module 4330 decompresses the test item at step S3750A and decryption module 4320 decrypts the test item at step S3765A. Request processor 4310 returns the test item to test driver application 3070 at step S3770A. However, if the test item is not available locally (e.g., miss), item request module 4350 requests the test item from item cache servers 3050 via Web service interface 4400 at steps S3725A and S3730A. Web service interface 4400 requests the test item from item cache server 3050 at step S3730A. Item cache server 3050 or the source server 3030 returns the requested test item at step S3735A. Authentication module 4410 authenticates the returned requested test item at step S3740A. Item cache controller 3120 requests storage instructions from cache controller 4400 at step 3745A and accordingly stores the returned requested test item in item cache 4360 at a step S3750A. Request processor 4310 retrieves the test item from item cache 4360 at step S3755A. If necessary, decompression module 4330 decompresses the test item at step S3760A and decryption module 4320 decrypts the test item at step S3765A. Request processor 4310 delivers the test item to test driver application 3070 at step S3770A.

[0236] A similar process occurs when test driver application 3070 requires a plugin. FIG. 37B is a flow chart of a method of caching plugins respect to the caching architecture according to the present invention. With reference to FIG. 37B, when test driver application 3070 requires a plugin, test driver application 3070 via plugin request interface 4370 instructs request processor 4310 to retrieve the plugin at steps S3710B and S3715B. Request processor 4310 verifies whether the plugin is available in plugin cache 4390 at step S3720B. If the requested plugin is available locally (e.g., hit), request processor 4310 retrieves the plugin from plugin cache 4390 at step S3755B. If necessary, decompression module 4330 decompresses the plugin at step S3760B and decryption module 4320 decrypts the plugin at step S3765B. Request processor 4310 returns the plugin to test driver application 3070 at step S3770B. However, if the plugin is not available locally (e.g., miss), plugin request module 4370 requests the plugin from plugin cache servers 3060 via Web service interface 4400 at steps S3725B and S3730B. Web service interface 4400 requests the plugin from plugin cache server 3060 at step S3730B. Plugin cache server 3060 or the source server 3030 returns the requested plugin at step S3735B. Authentication module 4410 authenticates the returned requested plugin at step S3740B. Plugin cache controller 3130 requests storage instructions from cache controller 4400 at step S3745A and accordingly stores the returned requested plugin in plugin cache 4390 at a step S3750B. Request processor 4310 retrieves the plugin from plugin cache 4390 at step S3755B. If necessary, decompression module 4330 decompresses the plugin at step S3760B and decryption module 4320 decrypts the plugin at step S3765B. Request processor 4310 delivers the plugin to test driver application 3070 at step S3770B. Moreover, a cache cleaning algorithm may be employed to retire test items and/or plugins when one or more predetermined conditions are satisfied, e.g., when a predetermined amount of time expires after the test item and/or the plugin is added to item cache 4360 or plugin cache 4390, respectively.

[0237] Caching of a computer-based test for delivery to candidate workstation 3000 is facilitated in accordance to the changing testing environment during computer-based testing reflected by monitoring of candidate progress, candidate performance, network bandwidth, network latency and server response, among other environmental variables, during computer-based testing. With reference to FIG. 36, test driver application 3070 further includes stimuli processor 4400, cache controller 4410, candidate performance monitor 4420, candidate progress monitor 4430, network latency monitor 4440, network bandwidth monitor 4450 and server response monitor 4460. In response to changes in the testing environment, stimuli processor 4410 adjusts the source for retrieving test components or the volume of test components to store in cache memory on candidate workstation 3000 and instructs cache controller 4410 accordingly. Environment monitoring means as shown in FIG. 36 include, but are not limited to, for example, candidate progress monitor 4430, which measures the rate at which the test candidate is answering test items; candidate performance monitor 4420, which measures test candidate competency; network bandwidth monitor 4450, which measures data transfer speed between candidate workstation 3000 and distributions servers; network latency monitor 4440, which measures delay times between candidate workstation 3000 and distributions servers caused by the network; and server response monitor 4460, which measures the delay times between candidate workstation 3000 and distributions servers when delay is caused by the server.

[0238] Descriptions of caching operations of a computer-based testing system according to the present invention is provided. Generally, stimuli processor 4410 periodically initiates an inquiry to candidate progress monitor 4430, candidate performance monitor 4420, network bandwidth monitor 4450, network latency monitor 4440 and server response monitor 4460 during computer-based testing. The results of each monitor are returned to stimuli processor 4410. Based these results, stimuli processor 4410 adjusts either the source of test components or the volume of test components being cached for delivery of the computer-based test and cache controller 4400 accordingly. Examples of the operations of the testing environment monitors candidate progress monitor 4430, candidate performance monitor 4420, network bandwidth monitor 4450, network latency monitor 4440 and server response monitor 4460 are now described.

[0239] For example, candidate progress monitor 4430 measures the test candidate's rate of progress in answering test items during computer-based testing for maintaining availability of test items. FIG. 39A shows a flow chart of the operation of candidate progress monitor 4430. Stimuli processor 4410 initiates an inquiry to candidate progress monitor 4430 at step S3800A. Candidate progress monitor 4430 retrieves the number of test items allotted to the current test section and the time allotted to each test item, which are stored in the exam resource file (described previously) on candidate workstation 3000 at step S3810A. Candidate progress monitor 4430 retrieves the number of test items answered by the test candidate and the answer time used by the test candidate for answering each test item, which are stored in the exam instance file (described previously) on candidate workstation 3000 at step S3815A. Candidate progress monitor 4430 then calculates the rate at which the test candidate answers test items at step S3820A and returns the number of test items allotted to the current test section, the number of test items answered by the test candidate in the current test section and the answer rate to stimuli processor 4410 at step S3825A for determining whether a sufficient pool of test items are available to the test candidate during computer-based testing. Based on these results, stimuli processor 4410 calculates the number of test items remaining in the current test section for delivery to the test candidate at S3830A. Stimuli processor 4410 then retrieves the number of test items stored in item cache 4360 at step S3835A and calculates the number of test items remaining to be cached in the current test section for delivery to the test candidate at step S3840A. Based on the answer rate and the number of test items remaining to be cached, stimuli processor 4410 determines whether it has become necessary to retrieve additional test items for caching at step S3845A. When it has become necessary to retrieve additional test items, stimuli processor 4410 instructs cache controller 4400 accordingly at step S3850A. Otherwise, monitoring continues at step S3855A.

[0240] In another example, candidate performance monitor 4420 measures the test candidate's competency in answering test items during computer-based testing for maintaining availability of suitable test items. FIG 39B shows a flow chart of the operation of candidate performance monitor 4420. Stimuli processor 4410 initiates an inquiry to candidate performance monitor 4420 at step S3800B. Candidate performance monitor 4420 retrieves the number of test items answered in the current test section, the competency level of each test item and the scoring of each test item, which are stored in the exam instance file (described previously) on the candidate workstation at step S3810B. Candidate performance monitor 4420 then calculates the competency level of the test candidate at step S3815B and returns the candidate competency level to stimuli processor 4410 at step S3820B for determining whether a sufficient pool of suitable test items are available to the test candidate during computer-based testing. Stimuli processor 4410 then retrieves the number of test items of the candidate competency level stored in item cache 4360 at step S3825B and calculates the number of test items of the candidate competency level remaining to be cached in the current test section for delivery to the test candidate at S3830B. Based on the candidate competency level and the number of test items of the candidate competency level remaining to be cached, stimuli processor 4410 determines whether it has become necessary to retrieve additional test items of the current competency level for caching at step

S3835B. When it has become necessary to retrieve additional test items of the candidate competency level, stimuli processor 4410 instructs cache controller 4400 accordingly at step S3840B. Otherwise, monitoring continues at step S3845B.

[0241] A further example is network bandwidth monitor 4450 which measures the speed of data transfer between candidate workstation 3000 and a distribution server for maintaining timely availability of test items. FIG. 39C shows a flow chart of the operation of network bandwidth monitor 4450. Stimuli processor 4410 initiates an inquiry to network bandwidth monitor 4450 at step S3800C. For each exchange between candidate workstation 3000 and a distribution server, network bandwidth monitor 4450 retrieves from logs generated by Web service interface 4400 the time a request is sent by candidate workstation 3000 to a distribution server, the time the request is received by the distribution server, the size of the request message, the time a response is sent by the answering distribution server, the time the response is received by candidate workstation 3000 and the size of the response message at step S3810C. Based on these variables, network bandwidth monitor 4450 calculates the data transfer speeds between candidate workstation 3000 and distribution servers at step S3015C and returns the data transfer speeds to stimuli processor 4410 at step S3820C for determining frequency of requests to retrieve test components and size of test components being transferred. Based on these results, stimuli processor 4410 calculates whether network bandwidth is growing or deteriorating at step S3825C and then stimuli processor 4410 instructs cache controller 4400 accordingly at step S3830C. For example, when network bandwidth is narrowing, frequency of data transfer may be increased and size of test components to be transferred may be decreased to compensate for loss of data transfer speed at step S3835C. Otherwise, monitoring continues at step S3840C.

[0242] An additional example, network latency monitor 4440 measures the delay in data transmission time between candidate workstation 3000 and the distribution servers caused by the network for re-establishing connectivity during computer-based testing. FIG. 39D shows a flow chart of the operation of network latency monitor 4440. Stimuli processor 4410 initiates an inquiry to network latency monitor 4440 at step S3800D. For each exchange between candidate workstation 3000 and a distribution server, network latency monitor 4440 retrieves from logs generated by Web service interface 4400 the time a request is sent by candidate workstation 3000 to a distribution server, the time the request is received by the distribution server, the size of the request message, the time a response is sent by the answering distribution server, the time the response is received by candidate workstation 3000, the size of the response message, the number of transmission retries and any network error messages received at step S3810D. Based on these variables, network latency monitor 4440 calculates delay times between candidate workstation 3000 and distribution servers caused by the network at step S3815D and returns the delay times to stimuli processor 4410 at step S3820D for determining the state of network connectivity. Stimuli processor 4410 retrieves a predetermined tolerance threshold from the exam resource file (described previously) at step S3825D and determines the rate delay times are approaching the tolerance threshold at step S3830D. Delay times approaching within a predetermined range of the tolerance threshold indicates that network connectivity failure is imminent. Prior to such failure, stimuli processor 4410 determines whether a sufficient number of test components are cached for delivery to the test candidate while the test driver reconnects to a redundant network in the background at step S3835D and instructs cache controller 4400 accordingly at step S3840D. Otherwise, monitoring continues at step S3845D.

[0243] In another example, server response monitor 4460 measures the delay in data transmission time between candidate workstation 3000 and the distribution servers caused by the distribution servers for maintaining accessibility to a source for test components during computer-based testing. FIG. 39E shows a flow diagram of the operation of server response monitor 4460. Stimuli processor 4410 initiates an inquiry to server response monitor 4460 at step S3800E. For each exchange between candidate workstation 3000 and a distribution server, server response monitor 4460 retrieves from logs generated by Web service interface 4400 the time a request from candidate workstation 3000 is received by a distribution server, time a response is sent by the answering distribution server, the number of retries and any server error messages received at step S3810E. Based on these variables, server response monitor 4460 calculates delay times between candidate workstation 3000 and a distribution server caused by the servers at step S3815E and returns the delay times to stimuli processor 4410 at step S3820E for determining the state of server accessibility. Stimuli processor 4410 retrieves a predetermined tolerance threshold from the exam resource file (described previously) at step S3825E and determines the rate delay times are approaching the tolerance threshold at step S3830E. Delay times approaching within a predetermined range of the tolerance threshold indicates that server accessibility is limited and stimuli processor 4410 instructs cache controller 4400 accordingly at step S3835E. Otherwise, monitoring continues at step S3840E.

[0244] The present invention is not limited to the embodiment described herein. For example, the test driver application and cacheable test components may be stored on the same distribution servers for delivery to test candidates. In addition, the distribution servers, in groups or singly, can be located in any number of remote locations. Furthermore, the testing environment monitors can include other monitors not specifically described herein. Thus, cacheable objects are used to expand functionality of a test driver application that controls delivery of a computer-based test to one or more test candidates over a dynamic distributed network environment by adapting delivery of the computer-based test in accordance to monitoring of testing environment variables.

[0245] The many features and advantages of the invention are apparent from the detailed specification, and thus, it is intended by the appended claims to cover all such features and advantages of the invention, which fall within the true spirit and scope of the invention. Further, since numerous modifications and variations will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction illustrated and described, and accordingly, all suitable modifications and equivalence may be resorted to, falling within the scope of the invention.

APPENDIX A-
UTD CLASSES AND INTERFACES

TABLE OF CONTENTS

TABLE OF CONTENTS 2

LOGICAL VIEW REPORT 4

LOGICAL VIEW 4
CComControl 4
CWindowImpl 4
DSADocs.DocDisplay 4
DSADocs.docItem 4
DSADocs.docReview 4
IAdviseSink 5
IDispatch 5
IOleClientSite 5
IOleDocumentImpl 5
IOleDocumentSite 5
IOleDocumentViewImpl 5
IOleInPlaceFrame 6
IOleInPlaceObjectWindowlessImpl 6
IOleInPlaceSite 6
IOleInPlaceUIWindow 7
IOleObject 7
IOleObjectImpl 8
IOleWindow 8
IPersistStorageImpl 8
UTDC.Browser 8
UTDC.GraphScores 10
UTDC.TimeRemaining 10
UTDComm.cCommentNavigation 10
UTDCore.Collection 10
UTDCore.ICompilerServices 10
UTDCore.cArea 11
UTDCore.cAttribute 11
UTDCore.cAttributes 11
UTDCore.cCategories 11
UTDCore.cCategory 12
UTDCore.cEvents 12
UTDCore.cExam 12
UTDCore.cForm 14
UTDCore.cGroup 15
UTDCore.cGroups 16
UTDCore.cItem 16
UTDCore.cItems 16
UTDCore.cMsgBox 17
UTDCore.cPresentation 17
UTDCore.cPresentations 18
UTDCore.cRegistry 18
UTDCore.cReport 19
UTDCore.cResults 19
UTDCore.cScript 19
UTDCore.cSection 20
UTDCore.cSections 21
UTDCore.eDirection 21

LOGICAL VIEW REPORT

UTDCore.ePluginModes ... *21*
UTDCore.eScoreStatus ... *21*
UTDCore.eStates ... *21*
UTDCore.iContainerNotify ... *22*
UTDCore.iContainerNotifyHelm ... *22*
UTDCore.iDisplay ... *22*
UTDCore.iHelm ... *22*
UTDCore.iItem ... *23*
UTDCore.iNavigate ... *23*
UTDCore.iPersistInstanceSet ... *24*
UTDCore.iPersistInstanceStore ... *24*
UTDCore.iPersistInstanceStream ... *24*
UTDCore.iPersistResourceSet ... *25*
UTDCore.iPersistResourceStore ... *25*
UTDCore.iPersistResourceStream ... *25*
UTDCore.iPlugin ... *25*
UTDCore.iReport ... *26*
UTDCore.iResults ... *26*
UTDCore.iScore ... *26*
UTDCore.iSelection ... *27*
UTDCore.iUnitTimer ... *27*
UTDP.StandardTimer ... *28*
UTDP.cBrowserDisplay ... *28*
UTDP.cLinearNavigate ... *28*
UTDP.cMutliChoiceItem ... *28*
UTDP.cNextPrevious ... *29*
UTDP.cSequentialExhastive ... *29*
UTDP.cSummaryScore ... *29*
UTDP.cUserSelection ... *29*
UTDPE.cChallenge ... *29*
UTDPE.cSLSDeliResult ... *29*
UTDPE.cStandardReview ... *29*
UTDPE.cStandardScoreReport ... *29*
UTDPE.cWTDResults ... *29*
cActivePlugin ... *29*
cContainerFrameT ... *29*
cDocumentSiteT ... *29*
cEvent ... *30*
cFormGroup ... *30*
cPlugInT ... *30*
cPlugin ... *30*
cScreenMinimum ... *30*
cTemplate ... *31*
iAppointment ... *31*
iLaunch ... *31*
iLaunch2 ... *31*
iPrint ... *31*
iTransfer ... *31*

TOTALS: ... 32

LOGICAL PACKAGE STRUCTURE ... 32

LOGICAL VIEW REPORT

Logical View

CComControl

Derived from CWindowImpl

Public Operations:

CComControl () : CComControl
FireOnRequestEdit (dispID : DISPID) : HRESULT
FireOnChanged (dispID : DISPID) : HRESULT
CreateControlWindow (hWndParent : HWND, rcPos : RECT&) : HWND
ControlQueryInterface (iid : const IID&, ppv : void) : HRESULT**

CWindowImpl

CWindowImpl allows you to create a new window or subclass an existing window

Public Attributes:

m_hWnd :

Public Operations:

Create () :
Creates a window
DefWindowProc () :
Provides default message processing

GetWndClassInfo () :
Returns a static instance of CWndClassInfo which manages the window class information
SubclassWindow () :
Subclasses a window

UnsubclassWindow () :
Restores a previously subclassed window
WindowProc () :
Processes messages sent to the window
GetWindowProc () :
Returns the current window procedure
GetCurrentMessage () :
Returns the current message
OnFinalMessage () :
Called after receiving the last message (typically WM_NCDESTROY)

DSADocs.DocDisplay

The DSA introduction and display screens. Modified to support the new interfaces.

DSADocs.docItem

The DSA multichoice item. Does graphic and text distracters. Handles voice-overs and BSL. Modified to support the new interfaces.

DSADocs.docReview

The DSA review screen. Modified to support the new interfaces.

IAdviseSink

Public Operations:

OnViewChange (dwAspect : DWORD, lindex : LONG) : void
Advises that view of object has changed
OnRename (pmk : IMoniker *) : void
Advises that name of object has changed
OnSave () : void
Advises that object has been saved to disk
OnClose () : void
Advises that object has been closed

IDispatch

Base class for all UTDCore COM Interfaces.

IOleClientSite

Public Operations:

SaveObject () : HRESULT
Saves embedded object.
GetMoniker (dwAssign : DWORD, dwWhichMoniker : DWORD, ppmk : IMoniker **) : HRESULT
Requests object's moniker
GetContainer (ppContainer : LPOLECONTAINER *) : HRESULT
Requests pointer to object's container
ShowObject () : HRESULT
Asks container to display object
OnShowWindow (fShow : BOOL) : HRESULT
Notifies container when object becomes visible or invisible
RequestNewObjectLayout () : HRESULT
Asks container to resize display site.

IOleDocumentImpl

Public Operations:

CreateView (pIPSite : IOleInPlaceSite *, pstm : IStream *, dwReserved : DWORD, ppView : IOleDocumentView **) :
EnumViews (ppEnum : IEnumOleDocumentViews, ppView : IOleDocumentView **) :**
GetDocMiscStatus (pdwStatus : DWORD *) :

IOleDocumentSite

Public Operations:

ActivateMe (pViewToActivate : IOleDocumentView *) : HRESULT
Activates an OLE Document Object

IOleDocumentViewImpl

Public Operations:

SetInPlaceSite (pIPSite : IOleInPlaceSite *) :
GetInPlaceSite (ppIPSite : IOleInPlaceSite **) :
GetDocument (ppunk : IUnknown **) :
SetRect (prcView : LPRECT) :
GetRect (prcView : LPRECT) :
SetRectComplex (prcView : LPRECT, prcHScroll : LPRECT, prcVScroll : LPRECT, prcSizeBox : LPRECT) :
Show (fShow : BOOL) :
UIActivate (fUIActivate : BOOL) :
Open () :
CloseView (dwReserved : DWORD) :
SaveViewState (pstm : LPSTREAM) :
ApplyViewState (pstm : LPSTREAM) :
Clone (pIPSiteNew : IOleInPlaceSite*, ppViewNew : IOleDocumentView) :**
ActiveXDocActivate (iVerb : LONG) :

IOleInPlaceFrame

Derived from IOleInPlaceUIWindow

Public Operations:

InsertMenus (hmenuShared : HMENU, lpMenuWidths : LPOLEMENUGROUPWIDTHS) : HRESULT
Allows container to insert menus
SetMenu (hmenuShared : HMENU, holemenu : HOLEMENU, hwndActiveObject : HWND) : HRESULT
Adds a composite menu to window frame
RemoveMenus (hmenuShared : HMENU) : HRESULT
Removes a container's menu elements
SetStatusText (pszStatusText : LPCOLESTR) : HRESULT
Sets and displays status text about
EnableModeless (fEnable : BOOL) : HRESULT
Enables or disables modeless dialog boxes
TranslateAccelerator (lpmsg : LPMSG, wID : WORD) : HRESULT
Translates keystrokes

IOleInPlaceObjectWindowlessImpl

//
IOleInPlaceObjectWindowlessImpl

Public Operations:

AddRef (: void) : ULONG
QueryInterface (riid : const IID&, ppvObject : void) : HRESULT**
Release (: void) : ULONG
UIDeactivate (: void) : HRESULT
GetWindow (phwnd : HWND*) : HRESULT
ContextSensitiveHelp (: BOOL) : HRESULT
InPlaceDeactivate (: void) : HRESULT
SetObjectRects (prcPos : LPCRECT, prcClip : LPCRECT) : HRESULT
ReactivateAndUndo (: void) : HRESULT
OnWindowMessage (msg : UINT, wParam : WPARAM, lParam : LPARAM, plResult : LRESULT*) : HRESULT
GetDropTarget (: IDropTarget) : HRESULT**

IOleInPlaceSite

Derived from IOleWindow

Public Operations:

CanInPlaceActivate () : HRESULT
Determines if the container can activate the object in place.
OnInPlaceActivate () : HRESULT
Notifies the container that one of its objects is being activated in place.
OnUIActivate () : HRESULT
Notifies the container that the object is about to be activated in place, and that the main menu will be replaced by a composite menu
GetWindowContext (ppFrame : IOleInPlaceFrame **, ppDoc : IOleInPlaceUIWindow **, lprcPosRect : LPRECT, lprcClipRect : LPRECT, lpFrameInfo : LPOLEINPLACEFRAMEINFO) : HRESULT
Enables an in-place object to retrieve window interfaces that form at the window object hierarchy, and the position in the parent window to locate the object's in-place activation window
Scroll (scrollExtent : SIZE) : HRESULT
Specifies the number of pixels by which the container is to scroll the object
OnUIDeactivate (fUndoable : BOOL) : HRESULT
Notifies the container to reinstall its user interface and take focus.
OnInPlaceDeactivate () : HRESULT
Notifies the container that the object is no longer active in place

DiscardUndoState () : HRESULT
Instructs the container to discard its undo state.
DeactivateAndUndo () : HRESULT
Deactivate the object and revert to undo state.
OnPosRectChange (lprcPosRect : LPCRECT) : HRESULT
Object's extents have changed

IOleInPlaceUIWindow

Derived from IOleWindow

Public Operations:

GetBorder (lprectBorder : LPRECT) : HRESULT
Specifies a RECT structure for toolbars and controls.
RequestBorderSpace (pborderwidths : LPCBORDERWIDTHS) : HRESULT
Determines if tools can be installed around object's window frame.
SetBorderSpace (pborderwidths : LPCBORDERWIDTHS) : HRESULT
Allocates space for the border.
SetActiveObject (pActiveObject : IOleInPlaceActiveObject *, pszObjName : LPCOLESTR) : HRESULT
Provides for direct communication between the object and each document and frame window.

IOleObject

Public Operations:

AddRef (: void) : ULONG
QueryInterface (riid : const IID&, ppvObject : void) : HRESULT**
Release (: void) : ULONG
SetExtent (dwDrawAspect : DWORD, psizel : SIZEL*) : HRESULT
GetExtent (dwDrawAspect : DWORD, psizel : SIZEL*) : HRESULT
SetClientSite (pClientSite : IOleClientSite*) : HRESULT
GetClientSite (ppClientSite : IOleClientSite) : HRESULT**
SetHostNames (: LPCOLESTR, : LPCOLESTR) : HRESULT
Close (dwSaveOption : DWORD) : HRESULT
SetMoniker (: DWORD, : IMoniker*) : HRESULT
GetMoniker (: DWORD, : DWORD, : IMoniker) : HRESULT**
InitFromData (: IDataObject*, : BOOL, : DWORD) : HRESULT
GetClipboardData (: DWORD, : IDataObject) : HRESULT**
DoVerb (iVerb : LONG, : LPMSG, : IOleClientSite*, : LONG, hwndParent : HWND, lprcPosRect : LPCRECT) : HRESULT
EnumVerbs (ppEnumOleVerb : IEnumOLEVERB) : HRESULT**
Update (: void) : HRESULT
IsUpToDate (: void) : HRESULT
GetUserClassID (pClsid : CLSID*) : HRESULT
GetUserType (dwFormOfType : DWORD, pszUserType : LPOLESTR*) : HRESULT
Advise (pAdvSink : IAdviseSink*, pdwConnection : DWORD*) : HRESULT
Unadvise (dwConnection : DWORD) : HRESULT
EnumAdvise (ppenumAdvise : IEnumSTATDATA) : HRESULT**
GetMiscStatus (dwAspect : DWORD, pdwStatus : DWORD*) : HRESULT
SetColorScheme (: LOGPALETTE*) : HRESULT

IOleObjectImpl

Public Operations:

AddRef (: void) : ULONG
QueryInterface (riid : const IID&, ppvObject : void) : HRESULT**
Release (: void) : ULONG
SetExtent (dwDrawAspect : DWORD, psizel : SIZEL*) : HRESULT
GetExtent (dwDrawAspect : DWORD, psizel : SIZEL*) : HRESULT
SetClientSite (pClientSite : IOleClientSite*) : HRESULT
GetClientSite (ppClientSite : IOleClientSite) : HRESULT**
SetHostNames (: LPCOLESTR, : LPCOLESTR) : HRESULT
Close (dwSaveOption : DWORD) : HRESULT
SetMoniker (: DWORD, : IMoniker*) : HRESULT
GetMoniker (: DWORD, : DWORD, : IMoniker) : HRESULT**
InitFromData (: IDataObject*, : BOOL, : DWORD) : HRESULT
GetClipboardData (: DWORD, : IDataObject) : HRESULT**
DoVerbPrimary (prcPosRect : LPCRECT, hwndParent : HWND) : HRESULT
DoVerbInPlaceActivate (prcPosRect : LPCRECT, : HWND) : HRESULT
DoVerbShow (prcPosRect : LPCRECT, : HWND) : HRESULT
DoVerbUIActivate (prcPosRect : LPCRECT, : HWND) : HRESULT
DoVerbHide (: LPCRECT, : HWND) : HRESULT
DoVerbOpen (: LPCRECT, : HWND) : HRESULT
DoVerbDiscardUndo (: LPCRECT, : HWND) : HRESULT
DoVerb (iVerb : LONG, : LPMSG, : IOleClientSite*, : LONG, hwndParent : HWND, lprcPosRect : LPCRECT) : HRESULT
EnumVerbs (ppEnumOleVerb : IEnumOLEVERB) : HRESULT**
Update (: void) : HRESULT
IsUpToDate (: void) : HRESULT
GetUserClassID (pClsid : CLSID*) : HRESULT
GetUserType (dwFormOfType : DWORD, pszUserType : LPOLESTR*) : HRESULT
Advise (pAdvSink : IAdviseSink*, pdwConnection : DWORD*) : HRESULT
Unadvise (dwConnection : DWORD) : HRESULT
EnumAdvise (ppenumAdvise : IEnumSTATDATA) : HRESULT**
GetMiscStatus (dwAspect : DWORD, pdwStatus : DWORD*) : HRESULT
SetColorScheme (: LOGPALETTE*) : HRESULT

IOleWindow

Public Operations:

GetWindow (phwnd : HWND *) : HRESULT
Gets a window handle.
ContextSensitiveHelp (fEnterMode : BOOL) : HRESULT
Controls enabling of context-sensitive help.

IPersistStorageImpl

Public Operations:

AddRef (: void) : ULONG
QueryInterface (riid : const IID&, ppvObject : void) : HRESULT**
Release (: void) : ULONG
GetClassID (pClassID : CLSID*) : HRESULT
IsDirty (: void) : HRESULT
Load (pStorage : IStorage*) : HRESULT
Save (pStorage : IStorage*, fSameAsLoad : BOOL) : HRESULT
InitNew (: IStorage*) : HRESULT
SaveCompleted (: IStorage*) : HRESULT
HandsOffStorage (: void) : HRESULT

Private Operations:

IPSI_GetIPersistStreamInit () : IPersistStreamInit*

UTDC.Browser

The UTD browser control is an Active X control that is a customized web browser control that is locked down for security reasons and supports scripting objects from UTDCore

Public Attributes:

oRegistry : IRegistry *
Returns the single instance of the registry object. Read-only.

Public Operations:

get_AllowPOLESS () :
gets AllowPOLESS flag
put_AllowPOLESS () :
Sets the AllowPOLESS flag
get_AllowFile () :
getsAllowFile flag
put_AllowFile () :
Sets the AllowFile flag

get_AllowURL () :
gets the AllowURL flag

put_AllowURL () :
Sets the AllowURL flag
get_RelativeFontSize () :
gets the relative font size
put_RelativeFontSize () :
sets the relaitive font size
LoadURL () :
browses to a URL from a string
Load () :
loads the browser from a stream
get_oExam () :
gets the oExam object
put_oExam () :
sets the oexam object
get_oSection () :
gets the oSection object

put_oSection () :
sets the oSection object
get_oPresentation () :
gets the opresentation object
put_oPresentation () :
sets the opresentation object
get_oItem () :
gets the oItem object
put_oItem () :
puts the oItem object
get_oForm () :
gets the oFrom object
put_oForm () :
sets the form object
get_oAppointment () :
gets the oAppointment object

put_oAppointment () :
sets the oAppointment object
get_ScrollBars () :
gets the Scrollbars flag

put_ScrollBars () :
sets the scrollbars
get_Document () :
Gets the document boject
Show () :
Makes the browser visible
Hide () :
Hides the browser control
Unload () :
Unloads the Browser control
Refresh () :
Refresh the page in the browser control

UTDC.GraphScores

The UTD sections scores control. This control displays bar-graph of a score or scores. This control will run if requested from HTML passed to the UTDC.Browser control.

These can be:
single score vs. passing,
Score by section,
or score per category.

Bruce

UTDC.TimeRemaining

This is a visible control showing the remaining time. It can be configured to show the reamining time for the section or exam or the minimum of both. It can be configured to flash or prompt with messages when time is expiring. The time points and the messages are configurable.

It should be scriptable from HTML.

UTDComm.cCommentNavigation

The navigation puts the test driver into comment period mode and re-navigates the sections also show any presentation in the comment period section.

UTDCore.Collection

Base UTD collection. All other collection classes follow this design.

Public Attributes:

Count : Long
Returns the number of members in a collection.
_NewEnum : IUnknown * *
Returns a Visual Basic enumerator object.

Public Operations:

Item (Index : Variant) : Variant
Returns a specific member of a Collection object either by position or key.

UTDCore.ICompilerServices

Provides nifty features to the plugin.

Public Attributes:

sResourceDirectory : BSTR
Get the directory of the resource file currently being compiled.
Read-only.
sSourceDirectory : BSTR
Get the directory of the source file currently being compiled.
Read-only.

Public Operations:

ParseHTML (sSourceFile : BSTR) : BSTR

Loads the source file into mshtml, and removes everything in the <body> except the specified labeled <div>.

C:\EXAM\ITEMBANK.HTM:
<div id="Schmackaroo">
... Some HTML here ...
</div>

Parse("C:\EXAM\ITEMBANK.HTM#Schmackaroo") returns "...Some HTML here..."

UTDCore.cArea

The area in the template for a plugin
There is no IArea interface.

Derived from cDocumentSiteT, UTDCore.iContainerNotify

Public Attributes:

bContained : Boolean

If TRUE the plugin is contained in the area as an active document. If FALSE it is a floating window. Usually full screen.

nTop : Integer

The top coordinate. Zero if not contained.

nLeft : Integer

The left coordinate. Zero if not contained.

nWidth : Integer

The width if the contiainer. Zero if not contained.

nHeigth : Integer

The heigth if the contiainer. Zero if not contained.

oPluginClass : cPlugin

The plugin class for this window.

oPluginActive : UTDCore.iPlugin

If the plugin class is active, this is the iPlugin interface to it.

sSize : String

The size of the area on the split axis. Can be a percentage, pixel count or *.

UTDCore.cAttribute

Public Attributes:

sName : BSTR

Get the attribute name string.

value : VARIANT

Get the attribute value.

UTDCore.cAttributes

Collection of UTDCore.cAttribute

Derived from UTDCore.Collection

UTDCore.cCategories

Collection of UTDCore.cCategory

Derived from UTDCore.Collection

UTDCore.cCategory

A category of sections or items. For example: "Reporting", "Objectivies", "Scoring"

Public Attributes:

sName : String
The name of the category.
bComplete : Boolean
If TRUE the category must contain all sections or items defined at a level.
bDuplicate : Boolean
If TRUE the category allows sections or items defined more than once at a level.
sDescription : String
The description of the category.
colMembers : UTDCore.cEvents
The collection of sections or items that apply to this category .
colSubCategories : UTDCore.cCategories
Collection of UTDCore.cCategory. (AKA sub-categories).
eContents : eCategoryContents
Read-only. The allowable types of members in the colMembers.

```
enum {
  eCategoryContent_Anything = 0,
  eCategoryContent_Sections = 1,
  eCategoryContent_Items = 2,
  eCategoryContent_Categories = 3,
} eCategoryContents;
```

Private Attributes:

oScore : UTDCore.iScore
The scoring plugin for the form.

UTDCore.cEvents

IEvents contains deliverable classes as IDispatch objects. They must be queried individually to determine their exact type.

IEvents can be:
cExam, cForm, cSection, cItem, cPresentation, cReport, cResults.

Derived from UTDCore.Collection

UTDCore.cExam

The exam instance. The root object of the driver exposed classes.

Derived from cContainerFrameT

Public Attributes:

sName : String
Read-only exam name.
sLanguage : String
The language of the exam content. Read-only.
sVersion : String
The version of the exam. Read-only.
oActiveForm : UTDCore.cForm
The current form. Only one form can be active for an exam.
eStatus : eExamStatus
The eExamStatus as follows:

eExam_Initializing
eExam_Starting
eExam_InProgress

eExam_Ended
eExam_Terminating

bShowResponseComment : Boolean
If TRUE the candidate should not see their response during the comment period.
colCustomAttributes : UTDCore.cAttributes
Collection of custom attributes. Read-only.
sTitle : String
Exam title. Read-only
colCategories : UTDCore.cCategories
The collection of all categories.
oResourceRoot : Object
The read-only reference to the root storage of the POLESS resource file. Interfaces: POLESS.cStorageRoot, iStorage, iRootStorage, iStorageVB.
oInstanceRoot : Object
The read-only reference to the root storage of the POLESS exam instance file. Interfaces: POLESS.cStorageRoot, iStorage, iRootStorage, iStorageVB.
colTemplates : UTDCore.cTemplates
Collection of all templates defined in the resource file.
Holds flat collection of all templates at Exam level
colItemsChosen : UTDCore.cItems
Return all items chosen by a selection plugin, at any level of the exam tree, in the order that they were chosen. These are items that have been delivered or will be delivered.
colAllSections : UTDCore.cSections
Return all sections chosen by a selection plugin, at any level of the exam tree, in the order that they were chosen. These are items that have been delivered or will be delivered.
oRegistry : UTDCore.cRegistry
Returns the single instance of the registry object. Read-only.

Private Attributes:

oFormGroup : cFormGroup
The form group to select from.

Public Operations:

Minimize () :
Minimumizes the main windows of UTD. Can be called be simulations to get UTD out of the way.
Restore () :
Restore the main windows of UTD. Can be called be simulations to restore UTDto the screen.
GetScript () : UTDCore.cScript
Returns a script object to be used by a plug-in.
SaveInstance () :
Save the state of the test driver to the instance file. All active objects are asked to serialize themselves to the instance file.

Any one with a reference to this object can initiate this process. It could be an item plugin that wants to save user interaction and state information.

The driver will save between each presentation.
GetTransfer () : ITransfer
Returns a UAS Transfer object to transfer results or other files to the data center.
The cExam object does not keep a reference to this object.
GetPrint () : IUASPrint
Returns a UAS print object to save for reprint score and other reports.
The cExam object does not keep a reference to this object.
FatalError (lNumber : Long, sSource : String, sDescription : String) :
Reports an error in a plugin to the test driver.

Normally errors will be reported back to the test driver via an HRESULT and supportting IErrorInfo. If the error was generated from something other than test driver initiated call, the plugin should report the error via this method.

An example usage could be the user clicks a button on your plugin item. The resulting actions generate an error.

Trace (lNumber : Long, sSource : String, sDescription : String) :

Reports an event in a plugin to the test driver.

Warn () :

Writes a 'Warning' level trace messate to the instance file.

Info () :

Writes a 'Information' level trace messate to the instance file.

Start () :

Tells the exam that the candidate has accept the exam and started. Normally called by the Navigation plugin in response to the eDirection_Start.

The administration system is informed that the exam has started and the candidate has accepted it.

Quit () :

Tells the exam that the candidate is quitting the exam. Normally called by the Navigation plugin in response to the eDirection_Quit. The navigation plugin should confirm this decision with the candidate.

This causes the test driver to stop delivery and shutdown execution cleanly. The administration system is informed that the candidate quit the exam.

GetMsgBox () : UTDCore.cMsgBox

Returns a message box object.

UTDCore.cForm

A exam form.

Derived from UTDCore.iContainerNotifyHelm

Public Attributes:

oCurSection : UTDCore.cSection

The current active section in the exam. Read-only.

colChildren : UTDCore.cEvents

The collection of all top level sections of the exam. Read-only.

colCustomAttributes : UTDCore.cAttributes

Collection of custom attributes. Read-only.

sName : String

Read-only form name.

sTitle : String

Form title. Read-only

colItemsChosen : UTDCore.cItems

The collection of all items chosen (that is, returned by a selection plugin) in the exam. This is regardless of their section.

colAllSections : UTDCore.cSections

The collection of all sections of the exam regardless of their level.

datStart : Date

The form start date and time.

datFinish : Date

The form finish date and time.

oTimer : UTDCore.iTimer

The timer for the form.

oScore : UTDCore.iScore

The scoring plugin for the form.

fTimerFactor : Single
The multiple for test timer based on ADA conditions.
sVersion : String
The version of the form.
colDelivered : UTDCore.cEvents
The collection of all delivered top level sections of the form. Read-only.
nCurChild : Long = 0
Index of last delivered event in colDelivered. Read-only.
eStatus : eScoreStatus
Returns the value of oForm.oScore.eStatus.
colAllPresentations : UTDCore.cPresentations
The collection of all presentations of the exam regardless of their level.
colPresentationsChosen : UTDCore.cPresentations
The collection of all presentations chosen (that is, returned by a selection plugin) in the exam. This is regardless of their section or group.
colSectionsChosen :
The collection of all sections chosen (that is, returned by a selection plugin) in the exam. This is regardless of their level.
colAllItems :
The collection of all items of the exam regardless of their level.
colAllGroups :
The collection of all groups of the exam regardless of their level.
colGroupsChosen :
The collection of all groups chosen (that is, the selection plugin for this group has been Reset()) in the exam. This is regardless of their level.

Public Operations:

SectionNext () :
Requests that the driver proceeds to the next section or score report or result writing or the comment period.
SectionPrevious () :
Requests that the driver proceeds to the previous section or score report or result writing or the comment period. This is often illegal and will not be allowed by the defined navigation plugin.
SectionGoto (vSection : Variant) :
Requests that the driver proceeds to a named or numbered section or score report or result writing or the comment period.
datMinRemaining () :
Minimum time remaining for this exam.

UTDCore.cGroup

Public Attributes:

sName : String
Read-only group name.
sTitle : String
Read-only group title.
colCustomAttributes : UTDCore.cAttributes
Collection of custom attributes. Read-only.
colChildren : UTDCore.cEvents
The collection of presentations, sub-sections, or sub-groups for this group.
oScoring : UTDCore.iScore
The scoring plugin for the section.
colDelivered :
Collection of presentations or sub-sections containing all delivered events within this group. This will not contain sub-groups.
oSelection : UTDCore.iSelection
Pointer to the current selection plugin of this group.
oSection : UTDCore.cSection
The section housing this group.

UTDCore.cGroups

Collection of UTDCore.cGroup

Derived from UTDCore.Collection

UTDCore.cItem

The item's general information. Stuff that is not plugin specific.
This class implements IcItem and IItem interface.

Derived from UTDCore.iItem

Public Attributes:

bPresented : Boolean = False
True if item has been presented to the candidate. Set by item component.

sName : String
Read-only item name.

sTitle : String
Presentation title. Read-only

oScriptStart : UTDCore.cScript
The script to execute before this object executes.

oScriptFinish : UTDCore.cScript
The script to execute after this object executes.

oScriptConditional : UTDCore.cScript
This expression script is evaluated. If TRUE this object is deliveried.

fWeight : Double = 1
The item weigthing relative to other items. Normally 1. If multiple weigths per item are required they can be attached in the custom attributes. The weigth can be negative.

colCustomAttributes : UTDCore.cAttributes
Collection of custom attributes. Read-only.

lSeconds : Long
The seconds spent on the item.

bScored : Boolean = TRUE
Read-only. TRUE if the item is to be scored.
If false fWeight and fScoreCanidateWeighted will all be 0.

bSkipAllowed : Boolean
Read-only property. If TRUE the item can be skipped. If TRUE the bProceed() method on iItem interface will not be called.

fScoreCanidateWeighted : Double
The weigthed value of the canidate for the item. The judged value for the item will by multiplied by the weight for this value.

sPluginName : String
Read-only plugin name. The prog ID of the plugin.

oSection : UTDCore.cSection
Read-only property of the section the item was deliveried in.

oPresentation : UTDCore.cPresentation
Read-only property of the presentation the item was deliveried in.

datElapsed : DATE
The time spent on the item. Same as lSeconds, but a different COM type.

UTDCore.cItems

Collection of UTDCore.cItem

Derived from UTDCore.Collection

UTDCore.cMsgBox

This class displays a message box with a candidate prompt. This class is returned from IExam::GetMsgBox(). The message in the box is supplied by the called. The title can optionally be supplied. The buttons are supplied by the caller, but the button labels are defined at the exam level in the resource file.

The parent of the message box is the main exam window. The message box can be set to time out after a duration.

Public Attributes:

sTitle : String = UTDCore

The title the message box will use. Read/Write.

lTimeout : Long = 120

The timeout value on the message box in seconds. Read/Write.
The Display will return eMsgBoxResult_Timeout.

nParentHWnd : Integer

The parent window handle Read/Write
Default is the main UTDCore window.

Public Operations:

Display (sPrompt : String, eStyle : eMsgBoxStyles) : eMsgBoxResult

Display the message box. Displays the buttons requested and the message supplied.

The return value is

```
enum {
  eMsgBoxResult_OK = 1,
  eMsgBoxResult_Cancel = 2,
  eMsgBoxResult_Abort = 3,
  eMsgBoxResult_Retry = 4,
  eMsgBoxResult_Ignore = 5,
  eMsgBoxResult_Yes = 6,
  eMsgBoxResult_No = 7,
  eMsgBoxResult_Timeout = 8
} eMsgBoxResults;
```

The style of message box.

```
enum {
  eMsgBoxStyle_OKOnly = 0,
  eMsgBoxStyle_OKCancel = 1,
  eMsgBoxStyle_AbortRetryIgnore = 2,
  eMsgBoxStyle_YesNoCancel = 3,
  eMsgBoxStyle_YesNo = 4,
  eMsgBoxStyle_RetryCancel = 5,
  eMsgBoxStyle_Critical = 16,
  eMsgBoxStyle_Question = 32,
  eMsgBoxStyle_Exclamation = 48,
  eMsgBoxStyle_Information = 64,
  eMsgBoxStyle_ApplicationModal = 0,
  eMsgBoxStyle_SystemModal = 4096,
} eMsgBoxStyles;
```

UTDCore.cPresentation

The data and current state of the presentation. Could be a item or a display.

Derived from cEvent, UTDCore.iContainerNotifyHelm

Public Attributes:

bCountPresentation : Boolean
If TRUE the presentation should count for the total X or Y for the section. "the item count"

colCustomAttributes : UTDCore.cAttributes
Collection of custom attributes. Read-only.

sName : String
Read-only presentation name.

sTitle : String
Presentation title. Read-only

oScriptStart : UTDCore.cScript
The script to execute before this object executes.

oScriptFinish : UTDCore.cScript
The script to execute after this object executes.

oScriptConditional : UTDCore.cScript
This expression script is evaluated. If TRUE this object is deliveried.

sComment : String
The user entered comment for this presentation.

bMarked : Boolean
TRUE if the item has been mark for later review. Set by the helm.

colChildren : UTDCore.cEvents
The items in this presentation.
The collection maybe empty.

bSelected : Boolean
True if the presentation or any of its items have been selected for delivery by a selection plugin.
Read-only.

Public Operations:

FindIcItem (pIItem : IItem *) : IcItem *
Find the IcItem interface for the given IItem.
Returns failure if the IItem is not in this presentation.

UTDCore.cPresentations

Collection of UTDCore.cPresentation

Derived from UTDCore.Collection

UTDCore.cRegistry

Alows registry look up for translating strings

For examples:

<base href="%REG:DSAGRAPGHICS">

into

<base href="V:\DSA\">

Looks at
[HKEY_LOCAL_MACHINE\SOFTWARE\Prometric\UTD\Keywords]

Public Operations:

Lookup (sKey : String) : String

Retrieves the string by finding the key in the registry and returning the value.

Translate (sInString : String) : String

Translates each occurance of %REG:x% into the equivilent of "x" in the registry.

UTDCore.cReport

A score report "section".

Derived from cEvent

Public Attributes:

oScoreReport : UTDCore.iScore

The score report plugin.

UTDCore.cResults

The results writing "section"

Derived from cEvent

Public Attributes:

oResults : UTDCore.iResults

The plugin that writes the results.

UTDCore.cScript

A script written in VBscript or JScript. To be executed or evaluated by the ActiveX Scripting Engine. This object will expose UTD objects to the script code. These objects will be based on the scope of the "Parent" object provided to this object.

All scripts can see oExam, oForm, and oCandidate.

If the parent object is a section object, then the script will also see oCurSection.

If the parent object is a presentation object, then the script will also see oCurSection and oCurPresentation.

If the parent object is an item then the script will also see oCurSection, oCurPresentation, and oCurItem.

Public Attributes:

oParent : Object

The parent UTD object. It could be a form, a section or a presentation.

Public Operations:

Execute (sScript : BSTR) :

Executes the script code passed in the sScript parameter.

EvaluateBool (sScript : BSTR) : Boolean

Evaluates the expression parsed as the sScript parameter. Returns TRUE or FALSE. Raises an error if the code is not an expression returning a boolean.

EvaluateVar (sScript : BSTR) : Variant

Evaluates the expression passed in the sScript parametert. Returns a variant depending on the return type of the script. Raises an error if the code is not an expression.

SetScope (oSection : ISection *, oPresentation : IPresentation *, oItem : IItem *) :

Set the parent section, presentation, and item. The script will have access to these as the oCurSection, oCurPresentation, and oCurItem objects. Any/All may be passed as NULL parameters indicating that the script does not have access to that particular object.

AddObject (sName : BSTR, pObject : IDispatch *) : HRESULT
Add an object into the global script scope.

Protected Operations:

LoadGlobalScript (oStream : POLESS.IcStorage) :
Called by the test driver to initialize the global script that is common to all later scripts. This will be shared by all instances of cScript.
The oStorage parameter is a storage containing an attribute set and a stream containing the global script.

UTDCore.cSection

Represents one exam sections or sub-section. May contain items or sections.
Derived from cEvent

Public Attributes:

oNavigation : UTDCore.iNavigate
The navigation plugin for this section.
colCustomAttributes : UTDCore.cAttributes
Collection of custom attributes. Read-only.
sName : String
Read-only section name.
sTitle : String
Section title. Read-only.
oScoring : UTDCore.iScore
The scoring plugin for the section.
oScriptStart : UTDCore.cScript
The script to execute before this object executes.
oScriptFinish : UTDCore.cScript
The script to execute after this object executes.
oScriptConditional : UTDCore.cScript
This expression script is evaluated. If TRUE this object is deliveried.
oReview : UTDCore.cPresentation
The review presentation for the section. It can be nothing if the section does not have a review defined for it.
oCurChild : Object
The current child being deliveried. This can be a cSection or a cPresentation.
datStart : Date
The section start date and time.
datFinish : Date
The section finish date and time.
colChildren : UTDCore.cEvents
The collection of presentations or sub-sections for the section.
oSelection : UTDCore.iSelection
The selection plugin for this section.
oTemplate : UTDCore.cTemplate
The tmeplate for the section.
oTimer : UTDCore.iTimer
The timer plugin for the section.
dtTimeRemaining : DATE
Returns the time remaining in the section. It get the seconds remaining from the iTimer plugin. It then converts it to a variant DATE type (aka VT_DATE).
colDelivered : UTDCore.cEvents
The collection of presentations or sub-sections for the section that have been presented. They are ordered by the order they appeared in.
bComment : Boolean = FALSE
The commenting mode of the section. Normally set by the Navigation.
bItemsReadOnly : Boolean = FALSE
The read-only mode of the items in the section. Normally set by the Navigation. Read only items cannot have their responses changed.

nCurChild : long = 0
Index of last delivered event in colDelivered. Read-only.

Public Operations:

ChildNext () :
Requests that the driver proceeds to the next presentation or child section.
ChildPrevious () :
Requests that the driver proceeds to the previous presentation or child section.
ChildGoto (vPresentation : Variant) :
Requests that the driver proceeds to a named or numbered presentation or child section..

UTDCore.cSections

Collection of UTDCore.cSection

Derived from UTDCore.Collection

UTDCore.eDirection

Enumeration of the possible directions to proceed from a presentation.
Values:
eDirection_Next
eDirection_Previous
eDirection_TimeoutImmedidate (now do not wait)
eDirection_TimeoutAfter (after current presentation)
eDirection_Review (Goto the review presentation)
eDirection_Jump (Goto a specific presentation)
eDirection_Flag
eDirection_Help
eDirection_FirstIncomplete
eDirection_FirstSkipped
eDirection_FirstPresentation

UTDCore.ePluginModes

Enumeration of the possible modes that an active document can behave.
Values:
ePluginMode_Author (Not supportted)
ePluginMode_Compile
ePluginMode_Delivery
ePluginMode_Rescore (Not suportted)
ePluginMode_ProcessResults (Not suportted)

UTDCore.eScoreStatus

The scoring status for a form/section/group/category, determined by the scoring plugin.

eScore_Incomplete
eScore_Complete
eScore_Failure
eScore_Pass
eScore_Taken

UTDCore.eStates

The states of the test driver.

eStateStart -- Initializing the test driver & reading the resource file.
eStateSectionDelivery -- Delivering a section.
eStateSectionComment -- Commenting all sections.

eStateSectionFeedBack -- Feedback on all sections.
eStateScoreReporting -- Rendering and printing a score report.
eStateResults -- Writing results.
eStateFinish -- Closing down the test driver.

UTDCore.iContainerNotify

All plugins receive a reference to this interface. It allow them to report events to the container.

Public Operations:

Activated () :

Call to indicate the plugin has received the presentation change and it fully ready for operations.

This includes: processing its data and if visible displaying.

UTDCore.iContainerNotifyHelm

This interface consumed by plugins to inform the container to navigate.

Public Operations:

RequestMove (eDirect : UTDCore.eDirection, sPresentation : String) :

Requests that the driver proceed in the direction specified. The driver next requests this movement from the navigation plugin.

The second parameter optional specifies the presentation. This is only used for the JUMP.

UTDCore.iDisplay

Interface supportted by plugins that a handle title bars, displays, non-answered items and summaries.

Public Attributes:

oSection : UTDCore.cSection

The section we are in.

Public Operations:

PresentationStarting (oPresentation : UTDCore.cPresentation, oContainerNotify : UTDCore.iContainerNotify) :

Called by the test driver to inform the plugin that a new presentation is starting. Also who to notify when active.

PresentationEnding () :

Called by the test driver to inform the plugin thae presentation is ending.

UTDCore.iHelm

Interface supportted by plugins that a handle navigation control or review. Like Next, Previous or Incomplete. Only User interface. Talks to the iNavigate plug in to perform the actual navigation.

Public Attributes:

oSection : UTDCore.cSection

The section we are in.

Public Operations:

PresentationStarting (oPresentation : UTDCore.cPresentation, oContainerNotifyHelm : UTDCore.iContainerNotifyHelm, oContainerNotify : UTDCore.iContainerNotify, bComment : Boolean) :

Called by the test driver to inform the plugin that a new presentation is starting. It also provides a way to notify the test driver of movement.

PresentationEnding () :

Called by the test driver to inform the plugin thae presentation is ending.

UTDCore.iItem

Interface supportted by plugins that a handle items and simulators.

Public Attributes:

oSection : UTDCore.cSection
The section we are in.

fScoreMinimum : Double = 0
The lowest possible score. Must be less than or equal to the nominal score. It can be negative.

fScoreNominal : Double = 0
The score if the candidate takes no actions. Must be less than or equal to the maximum score. Must be greater or equal to the minimum score. Normally zero.

fScoreMaximum : Double = 1
The highest possible score. Must be greater than or equal to the nominal score.

fScoreCandidate : Double = 0
The score received by the candidate. Requests that the plugin judges the item.

bComplete : Boolean = False
True if Item is complete.

bSkipped : Boolean = False
True if item was skipped by the candidate.

Public Operations:

bProceed (eDirect : UTDCore.eDirection) : Boolean
Test Driver will call this method to see if it is OK to move off the current presentation. The parameter indicates the direction.

GetCorrectAnswerDisplay () : String
Returns the correct answer in a form to be displayed on the screen.

GetCorrectAnswerResults () : Byte()
Returns the correct answer in a form to be written to the results file. The correct answer is in the form of a safe array of bytes.

GetResponseDisplay () : String
Returns the candidate response in a form to be displayed on the screen.

GetResponseResults () : Byte()
Returns the candidate response in a form to be written to the results file. The response is in the form of a safe array of bytes.

PresentationStarting (oPresentation : UTDCore.cPresentation, oContainerNotify : UTDCore.iContainerNotify, bReadOnly : Boolean) :
Called by the test driver to inform the plugin that a new presentation is starting.

PresentationEnding () :
Called by the test driver to inform the plugin thae presentation is ending.

UTDCore.iNavigate

Interface supportted by plugins that a handle section navigation.

Implementer of iNavigate must also implement a iContainerNotifyHelm

Derived from UTDCore.iContainerNotifyHelm

Public Attributes:

nPresentations : Integer
The number of counted presented to be presented within the scope of the navigation. Each item in a compound items count as a one.

bItemsDetermined : Boolean
TRUE if the number of items to present within the scope of the navigation can be determined. Adaptive items or sections can prevent this.

Public Operations:

bNavigateAllowed (eDirect : UTDCore.eDirection) : Boolean

Returns TRUE if navigation in the direction supplied sis allowed. It is the helm's job to query this before every presentation.

This is a combination of the resource file plugin data and the current state of the section.

Example implementation:
The test designer is allowed to configure if the prévisous button appears. But if we are on the first presentation of a section it will not appear even if configured.

Starting (oSection : UTDCore.cSection) :

The section is starting.

Ending () :

The section is ending.

UTDCore.iPersistInstanceSet

Interface for a plugin that wants to be persistant in the instance file as a property set.

Public Attributes:

IsDirty : Boolean

Returns TRUE if the object needs to save state to the instance file.

Public Operations:

Save (oPropSet : POLESS.IPropertyStorageVB) :

Called by the test driver to save the plugin data to the instance file.

Reload (oPropSet : POLESS.IPropertyStorageVB) :

Called by the test driver to reload the plugin data from the instance file. The IPersistResource*::Load will be called next.

UTDCore.iPersistInstanceStore

Interface for a plugin that wants to be persistant in the instance file as a storage.

Public Attributes:

IsDirty : Boolean

Returns TRUE if the object needs to save state to the instance file.

Public Operations:

Save (oStorage : POLESS.iStorageVB) :

Called by the test driver to save the plugin data to the instance file.

Reload (oStorage : POLESS.iStorageVB) :

Called by the test driver to reload the plugin data from the instance file. The IPersistResource*::Load will be called next.

UTDCore.iPersistInstanceStream

Interface for a plugin that wants to be persistant in the instance file as a property stream.

Public Attributes:

IsDirty : Boolean

Returns TRUE if the object needs to save state to the instance file.

Public Operations:

Save (oStream : POLESS.IStreamVB) :

Called by the test driver to save the plugin data to the instance file.

Reload (oStream : POLESS.IStreamVB) :

Called by the test driver to reload the plugin data from the instance file. The IPersistResource*::Load will be called next.

UTDCore.iPersistResourceSet

Interface for a plugin that wants to be persistant in the resource file as a property set.

Public Operations:

Save (oPropSet : POLESS.IPropertyStorageVB) :
Called by the compiler to save the plugin data to the resource file.

Load (oPropSet : POLESS.IPropertyStorageVB) :
Called by the test driver to load the plugin data from the resource file.

ValidateResource (oPropSet : POLESS.IPropertyStorageVB, oCompilerServices : UTDCore.ICompilerServices) :
Validates the compiled resource for the plugin. The plugin must raise an automation error that describes any problems with the source.

The combination of the amalogmated storage for the XXL and the referenced files should resolve all references. For example all HREFs to HTML and JPEGs should resolve.

UTDCore.iPersistResourceStore

Interface for a plugin that wants to be persistant in the resource file as a storage.

Public Operations:

Save (oStorage : POLESS.iStorageVB) :
Called by the compiler to save the plugin data to the resource file.

Load (oStorage : POLESS.IStorageVB) :
Called by the test driver to load the plugin data from the resource file.

ValidateResource (oStorage : POLESS.iStorageVB, oCompilerServices : UTDCore.ICompilerServices) :
Validates the compiled resource for the plugin. The plugin must raise an automation error that describes any problems with the source.

The combination of the amalogmated storage for the XXL and the referenced files should resolve all references. For example all HREFs to HTML and JPEGs should resolve.

UTDCore.iPersistResourceStream

Interface for a plugin that wants to be persistant in the resource file as a stream.

Public Operations:

Save (oStream : POLESS.IStreamVB) :
Called by the compiler to save the plugin data to the resource file.

Load (oStream : POLESS.IStreamVB) :
Called by the test driver to load the plugin data from the resource file.

UTDCore.iPlugin

This Interface will be supportted by all UTD plugins.

All plugins must implement the following interfaces:
iPlugin

All plugin must implement one of the following to persist into the resource file at compile time:
IPersistResourceSet, IPersistResourceStore or IPersistResourceStream

All plugin may implement one of the following to persist into the instance file during delivery time:
IPersistInstanceSet, IPersistInstanceStore or IPersistInstanceStream

if you are visible one of these: iHelm, iItem or iDisplay. It can be an iHelm and an iItem. If you wish to be contained as an active document you must support the following: iDataObject, IoleInPlaceObject, IoleInPlaceActiveObject , IoleDocument, IOleDocumentView and IoleCommandTarget.

or one of these if you are invisible:
iScore, iReport, iResults, iNavigate, iTimer.

iNavigate must also implement a iContainerNotifyHelm

Public Attributes:

eMode : ePluginModes
The mode the plugin should operate in.
Set by the consumer of the plugin (the driver or compiler).

Public Operations:

ValidateSource (oSource : POLESS.IStreamVB, oCompilerServices : UTDCore.ICompilerServices) :
Validates the source XXL for the plugin. The plugin must raise an automation error that describes any problems with the source. The source is not required to be complete only the portions provided should be verified.

If the contents of the stream is Unicode it will be marked with the BOM (byte order mark) as defined be unicode standard (www.unicode.org). The BOM is normally FFFE.

If the stream contains ASCII or UTF-8, no BOM will be included.

The oCompilerServices interface is provided to offer additional features and information to the plugins.

Unload () :
Unload data and references to UTD objects.

Load (oExam : UTDCore.cExam, oCandidate : iAttendance) :
Load with references to UTD objects. Only called during exam delivery.

UTDCore.iReport

Interface supported by plugins that handle printing of the score reports and other other material like stylus ("hand-outs").

Public Operations:

PrintReport () :
Prints the score report

UTDCore.iResults

Interface supported by plugins that handles writing of the candidate results.

Public Operations:

WriteResults () :
Write the exam results.

UTDCore.iScore

Interface supportted by plugins that a handle scoring of sections or exams.

Public Attributes:

eStatus : UTDCore.eScoreStatus
The eScoreStatus as follows.

fReportedScoreCut : Double = -1
The cut score to report to the candidate and results. Usually scaled. If this does not apply to the plugin return -1.

fReportedScoreMax : Double = -1
The maximum score to report to the candidate and results. Usually scaled. If this does not apply to the plugin return -1.

fReportedScoreCandidate : Double = -1
The achivied score to report to the candidate. Usually scaled. If this does not apply to the plugin return -1.

oEvents : IEvents *

Tthe collection of children to score. If the plugin is on a section, then this collection can include ISection and IPresentation objects. If the plugin is on a category, then this collection can include ISection, and IItem objects.

Public Operations:

bStop () : Boolean

Test Driver will call this method to see if delivery should be stopped from current delivery unit (section or form).

For example an adaptive exam may return false when it is clear if the candidate has master or not mastered the material.

GetScoreDisplay () : String

Returns the score in a form to be displayed on the screen.

GetScoreResults () : Byte()

Returns the score in a form to be written to the results file. The score is in the form of a safe array of bytes.

UTDCore.iSelection

Selects items, sections or forms from a category.

Public Attributes:

bMore : Boolean

Returns TRUE if there are more objects to get. Read-only.

nCountTotal : Integer

Returns the total count of objects (items, sections or forms) that it has in its pool to select from. It will return -1 if this is not quantifiable.

nCountSelected : Integer

Returns the selected count of objects (items, sections or forms) that it will return from the pool. It will return -1 if this is not quantifiable. Like items in an adaptive section.

oGroup : UTDCore.cSection

Read-write.

The group that contains this selection plugin. This is not set for a form selection plugin.

oEvents : UTDCore.IEvents

Read-write.

The events collection to select from.

Public Operations:

Reset () :

This gives the selection plug a chance to prepare the data for the first selection. Some selection plugins like "random/exshastive" make pre-determine all the items for selection a store them in its internal memory and the instance file.

GetNext () : Object

Gets the next item, section or form. If there are no more, them NULL AKA Nothing is returned.

UTDCore.iUnitTimer

A plugin to perform timing across testing units.

Public Attributes:

lSecondsElasped : Long

The seconds elasped.

lSecondsRemaining : Long

The seconds remaining. If the unit is not timed, this will return -1.

lSecondsTotal : Long

The total seconds available for the unit.

datElasped : DATE

The time elasped. Same as lSecondsElasped.

datRemaining : DATE
The time remaining. If the unit is not timed, this will return -1. Same as lSecondsRemaining
datTotal : DATE
The total time available for the unit. Same as lSecondsTotal.

Public Operations:

PresentationActive (oPresentation : UTDCore.cPresentation) :
This is called by the test driver to inform the timer that the presentation is completely active. All plugins have loaded their data, displayed and are operational.
PresentationInactive () :
This is called by the test driver to inform the timer that the presentation is about to become inactive. The user has selected a direction to go from the item. The Item and the navigation has cleared it for movement.

After this call the plugins will be hidden and unloaded.
AddTime (nMinutes : Integer) :
Adds time to the current form or section. Time is specified in minutes.
Starting (oChildren : UTDCore.cEvents, oContainerNotifyHelm : UTDCore.iContainerNotifyHelm) :
Informs the plugin that the section or form is starting. If the plugin is defined at the form level then this is called when the form starts. If the plugin is defined at the section level then this is called when the section starts.

It gives it the collection of children to time across. This collection can include cSection objects and cPresentation objects.
Ending () :
Informs the plugin that the section or form is ending. If the plugin is defined at the form level then this is called when the form ends. If the plugin is defined at the section level then this is called when the section ends.

Pause () :
Pauses the timer
Resume () :
Resumes the timer. (After a Pause())

UTDP.StandardTimer
Standard timing pluggin works in three basic modes:

Non-timing, the unit's time does not contribute
Timed, the unit is timed and will timeout
Contributing, the unit time is contributed to the parent unit.

In the case of timeout it will be handled mroe than one ways:
Immediate end of unit,
End after current presentation,
Warnings.

UTDP.cBrowserDisplay
This display plugin renders HTML to the full area of the active document. It also interpets scripts in the HTML. The active object model of UTD is exposed to the script.

UTDP.cLinearNavigate
Performs linear navigation through a sections until the items or sections are exshasted.

UTDP.cMutliChoiceItem
The standard multichoice item. Uses HTML presentation layout.

UTDP.cNextPrevious
Standard Next/Previous type "toolbar" helm.

UTDP.cSequentialExhastive
This plugin performs sequenial exahstive selections. All objects (sections or items) in the order from the resource file.

UTDP.cSummaryScore
This plug performs scoring by summarizing items or section scores. The scores it summarizes can be weighted. The overall score can be scaled.

UTDP.cUserSelection
Allows a user to select a section from a list. The list is the sub-sections in the section.

UTDPE.cChallenge
This display is used to challenge the user to identify themselves. It is used to introduce the exam and during a break.

UTDPE.cSLSDeliResult
This result processor creates results for the SLSDC result process that looks like an SLSDeli result.

UTDPE.cStandardReview
This is the standard review screen. It displays all the items and their titles. It allows the candidate to:
Jump to any item
Goto the first incomplete
Goto the first skipped
Goto the first flags
End the section.

UTDPE.cStandardScoreReport
The standard score report. Will print an HTML page.

Public Operations:

get_AllowPOLESS () :
returns AllowPOLESS flag
putAllowPOLESS () :

UTDPE.cWTDResults
Writes a standard WTD/TASC results file.

cActivePlugin
Helper class for plugins.

cContainerFrameT
Template class for a container frame. (The main program window.)

Derived from IOleInPlaceFrame, CWindowImpl

Public Operations:

Create () :
Destroy () :

cDocumentSiteT
Template class for a document site. (Child window of the main program window.)

Derived from IOleInPlaceSite, IOleDocumentSite, IAdviseSink, IOleClientSite, CWindowImpl, IOleInPlaceUIWindow

Public Operations:

Create () :
Destroy () :

cEvent

The base class is used for all events deliveried in the exam. Like: sections, reports, results & presentations.

cFormGroup

The contain a from group. Used to select the form.

Public Attributes:

colForm : UTDCore.cEvents
Collection of UTDCore.cForm objects.
oForm : UTDCore.cForm
The selected form.

Private Attributes:

oSelection : UTDCore.iSelection
The selection plugin to use to select the form.

cPlugInT

Template class for a plug in. Gives active document functionality

Derived from IOleObjectImpl, IOleInPlaceObjectWindowlessImpl, IPersistStorageImpl, CComControl, IOleDocumentImpl, IOleDocumentViewImpl

cPlugin

This class holds the plugin information.

Public Attributes:

sName : String
Read-only name of the plugin XXL name. (read-only)
sProgID : String
The ProgID of the plugin. (read-only)
guidCLSID : GUID
The class ID of the plugin (read-only).

Public Operations:

Create () : UTDCore.iPlugin
CoCreates an instance of the plugin and returns it to the caller. This class does not retain any reference to this instance. The caller must release it.

cScreenMinimum

The minimum screen resolution

Public Attributes:

nWidth : Long = 800
The minimum screen width.
nHeigth : Long = 600
The minimum screen heigth.
nBitDepth : Integer = 8
The minimum screen color bit depth. 4 = 16 colors, 8 = 256 colors, 16 = 65768 colors, etc..

bValid : Boolean
If TRUE then current video configuration meets the mimimum requirements.

cTemplate

A template that defines the presentation areas on the screen. Referenced by a presentation.

Public Attributes:

sName : String
Read-only presentation template name.
colAreas : UTDCore.cAreas
Collection of areas in the template.
colTemplates : UTDCore.cTemplates
Nested templates for THIS template.

Public Operations:

Activate () :
Active this template. All areas instanciate their plugins and contain them. Not available outside the driver.
Deativate () :
Deactive this template. All areas destroy their plugins. Not available outside the driver.

iAppointment

This interface is part of the Unified Administration System. It allows access to the candidate information for the candidate taking this exam.

It is also emulated by the UTDTOETS component that emulates the UAS.

iLaunch

The interface to the DSA Administration system. It is how the admiinistration system controls a component in the center.

iLaunch2

The interface to the Unified Administration system. It is how the admiinistration system controls a component in the center.

iPrint

The administration system interface for save score reports for reprint. Also handles initial printing.

iTransfer

Administration interface to transfer results and other files back to the data center. It includes routing.

TOTALS:

2 Logical Packages
95 Classes

LOGICAL PACKAGE STRUCTURE

Logical View

Table of Contents

SELECTED COMPONENT VIEW REPORT 4

COMPONENT VIEW 4

UTDCore *4*

UTDCore.eMsgBoxResults (Interfaces) 5
UTDCore.IPresentation (Interfaces) 6
UTDCore.IContainerNotify (Interfaces) 7
UTDCore.ICategories (Interfaces) 8
UTDCore.IMsgBox (Interfaces) 8
UTDCore.IPersistResourceSet (Interfaces) 9
UTDCore.IGroup (Interfaces) 9
UTDCore.IScript (Interfaces) 10
UTDCore.IUnitTimer (Interfaces) 12
UTDCore.INavigate (Interfaces) 13
UTDCore.IItems (Interfaces) 15
UTDCore.ICategory (Interfaces) 15
UTDCore.IPersistInstanceStore (Interfaces) 16
UTDCore.IcResults (Interfaces) 16
UTDCore.IcItem (Interfaces) 17
UTDCore.IScore (Interfaces) 19
UTDCore.IExam (Interfaces) 20
UTDCore.IRegistry (Interfaces) 23
UTDCore.IAttribute (Interfaces) 24
UTDCore.IHelm (Interfaces) 24
UTDCore.IcReport (Interfaces) 25
UTDCore.eDirection (Interfaces) 25
UTDCore.IPersistInstanceStream (Interfaces) 26
UTDCore.ePluginModes (Interfaces) 26
UTDCore.IPersistResourceStream (Interfaces) 27
UTDCore.ISelection (Interfaces) 27
UTDCore.IGroups (Interfaces) 28
UTDCore.Collection (Interfaces) 28
UTDCore.eScoreStatus (Interfaces) 29
UTDCore.eStates (Interfaces) 29
UTDCore.IPersistResourceStore (Interfaces) 29
UTDCore.eExamStatus (Interfaces) 30
UTDCore.IItem (Interfaces) 30
UTDCore.IPlugin (Interfaces) 32
UTDCore.IContainerNotifyHelm (Interfaces) 33
UTDCore.IReport (Interfaces) 34
UTDCore.IForm (Interfaces) 34
UTDCore.IAttributes (Interfaces) 37
UTDCore.IPresentations (Interfaces) 37
UTDCore.ISections (Interfaces) 37
UTDCore.IDisplay (Interfaces) 37
UTDCore.ISection (Interfaces) 38
UTDCore.IResults (Interfaces) 41
UTDCore.eCategoryContent (Interfaces) 41
UTDCore.IEvents (Interfaces) 41
UTDCore.IPersistInstanceSet (Interfaces) 42

Administration System *42*

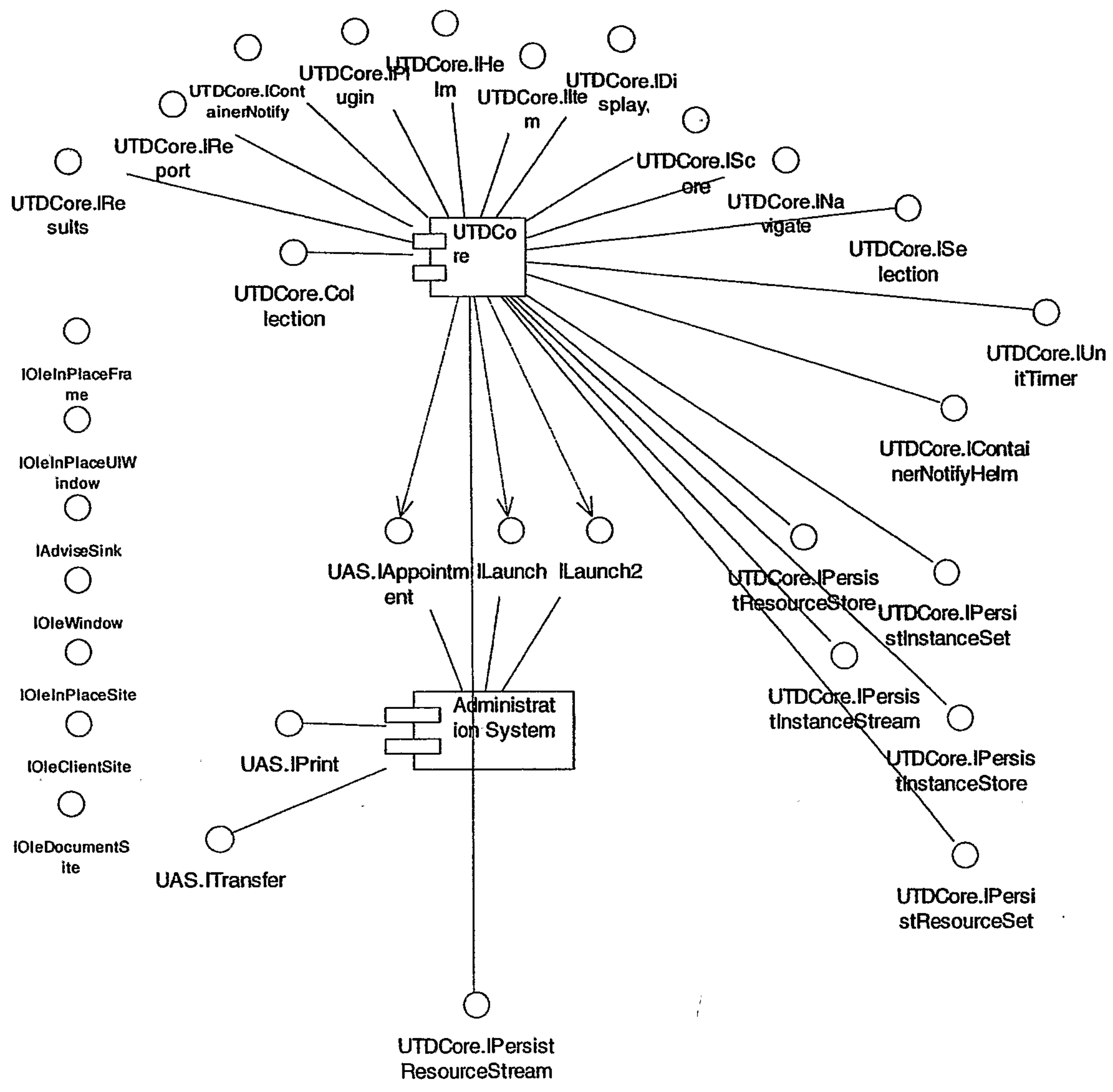
UAS.ITransfer (Interfaces) 42
UAS.IPrint (Interfaces) 42
UAS.IAppointment (Interfaces) 43

TOTALS: **44**

COMPONENT PACKAGE STRUCTURE 44

PHYSICAL VIEW REPORT

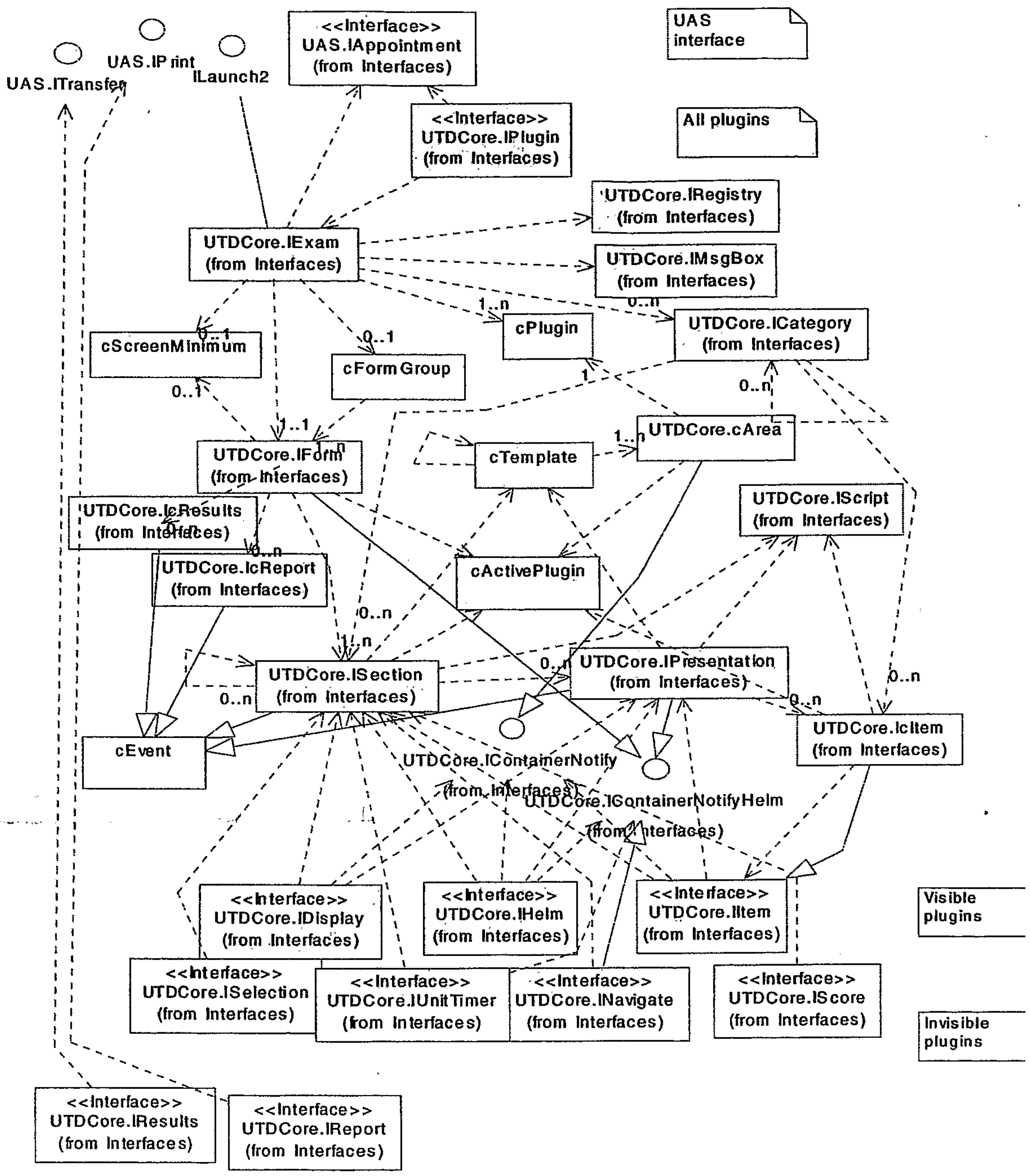
SELECTED COMPONENT VIEW REPORT



Component View

UTDCore

PHYSICAL VIEW REPORT


UAS.ITransfer
UAS.IPrint
ILaunch2
<<Interface>> UAS.IAppointment (from Interfaces)
UAS interface
<<Interface>> UTDCore.IPlugin (from Interfaces)
All plugins
UTDCore.IRegistry (from Interfaces)
UTDCore.IExam (from Interfaces)
UTDCore.IMsgBox (from Interfaces)
1..n
0..n
cPlugin
UTDCore.ICategory (from Interfaces)
cScreenMinimum
0..1
cFormGroup
0..1
0..1
1
0..n
1..1
UTDCore.IForm (from Interfaces)
1..n
UTDCore.cArea
cTemplate
UTDCore.IcResults (from Interfaces)
0..n
UTDCore.IScript (from Interfaces)
UTDCore.IcReport (from Interfaces)
cActivePlugin
0..n
1..n
UTDCore.ISection (from Interfaces)
0..n
0..n
UTDCore.IPresentation (from Interfaces)
0..n
0..n
cEvent
UTDCore.IcItem (from Interfaces)
UTDCore.IContainerNotify (from Interfaces)
UTDCore.IContainerNotifyHelm (from Interfaces)
<<Interface>> UTDCore.IDisplay (from Interfaces)
<<Interface>> UTDCore.IHelm (from Interfaces)
<<Interface>> UTDCore.IItem (from Interfaces)
Visible plugins
<<Interface>> UTDCore.ISelection (from Interfaces)
<<Interface>> UTDCore.IUnitTimer (from Interfaces)
<<Interface>> UTDCore.INavigate (from Interfaces)
<<Interface>> UTDCore.IScore (from Interfaces)
Invisible plugins
<<Interface>> UTDCore.IResults (from Interfaces)
<<Interface>> UTDCore.IReport (from Interfaces)

UTD version 2

UTDCore.eMsgBoxResults (Interfaces)

Enumeration of possible results from IMsgBox::Display

eMsgBoxResult_Yes
eMsgBoxResult_No
eMsgBoxResult_Timeout
(Candidate clicked nothing, timeout occurred)

UTDCore.IPresentation (Interfaces)

The data and current state of the presentation. Could be a item or a display.

Derived from cEvent, UTDCore.IContainerNotifyHelm

Public Attributes:

bCountPresentation : Boolean

If TRUE the presentation should count for the total X or Y for the section. "the item count"

colCustomAttributes : UTDCore.IAttributes

Collection of custom attributes. Read-only.

sName : String

Read-only presentation name.

sTitle : String

Presentation title. Read-only

oScriptStart : UTDCore.IScript

The script to execute before this object executes.

oScriptFinish : UTDCore.IScript

The script to execute after this object executes.

oScriptConditional : UTDCore.IScript

This expression script is evaluated. If TRUE this object is deliveried.

sComment : String

The user entered comment for this presentation.

bMarked : Boolean

TRUE if the item has been mark for later review. Set by the helm.

colChildren : UTDCore.IEvents

The items in this presentation. The collection will be empty for a non-item presentation.

bSelected : Boolean

True if the presentation or any of its items have been selected for delivery by a selection plugin. Read-only.

Public Operations:

FindIcItem (pIItem : IItem *) : IcItem *

Find the IcItem interface for the given IItem.
Returns failure if the IItem is not in this presentation.

UTDCore.IContainerNotify (Interfaces)

All plugins receive a reference to this interface. It allow them to report events to the container.

Public Operations:

Activated () :

Call to indicate the plugin has received the presentation change and it fully ready for operations. This includes processing its data and painting on the screen (if visible). This object is passed as a parameter on the PresentationStarting method of visible plug-in types.

Failure to call this method from a visible plug-in will cause the driver to wait infinitely. The candidate will not be able to navigate or respond. Do not call this method more than once for each time PresentationStarting is called.

UTDCore.ICategories (Interfaces)

Collection of UTDCore.ICategory

Derived from UTDCore.Collection

UTDCore.IMsgBox (Interfaces)

This class displays a message box with a candidate prompt. This class is returned from IExam::GetMsgBox(). The message in the box is supplied by the called. The title can optionally be supplied. The buttons are supplied by the caller, but the button labels are defined at the exam level in the resource file.

The parent of the message box is the main exam window. The message box can be set to time out after a duration.

Public Attributes:

sTitle : String = UTDCore

The title the message box will use. Read/Write.

lTimeout : Long = 120

The timeout value on the message box in seconds. Read/Write.

The Display will return eMsgBoxResult_Timeout.

nParentHWnd : Integer

The parent window handle Read/Write

Default is the main UTDCore window.

Public Operations:

Display (sPrompt : String, eStyle : UTDCore.eMsgBoxStyle) : UTDCore.eMsgBoxResults

Display the message box. Displays the buttons requested and the message supplied.

UTDCore.IPersistResourceSet (Interfaces)

Interface for a plugin that wants to be persistant in the resource file as a property set.

Public Operations:

Save (oPropSet : POLESS.IPropertyStorageVB) :

Called by the compiler to save the plugin data to the resource file.

Load (oPropSet : POLESS.IPropertyStorageVB) :

Called by the test driver to load the plugin data from the resource file.

ValidateResource (oPropSet : POLESS.IPropertyStorageVB, oCompilerServices : UTDCore.ICompilerServices) :

Validates the compiled resource for the plugin. The plugin must raise an automation error that describes any problems with the source.

The combination of the amalogmated storage for the XXL and the referenced files should resolve all references. For example all HREFs to HTML and JPEGs should resolve.

UTDCore.IGroup (Interfaces)

Groups are the holders for events in the section. A group may hold presentations or sections.

Public Attributes:

sName : String

Read-only group name.

sTitle : String

Read-only group title.

colCustomAttributes : UTDCore.IAttributes

Collection of custom attributes. Read-only.

colChildren : UTDCore.IEvents

The collection of presentations, sub-sections, or sub-groups for this group.

oScoring : UTDCore.IScore

The scoring plugin for the section.

colDelivered :

Collection of presentations or sub-sections containing all delivered events within this group. This will not contain sub-groups.

oSelection : UTDCore.ISelection

Pointer to the current selection plugin of this group.

oSection : UTDCore.ISection

The section housing this group.

UTDCore.IScript (Interfaces)

A script written in VBscript or JScript. To be executed or evaluated by the ActiveX Scripting Engine. This object will expose UTD objects to the script code. These objects will be based on the scope of the "Parent" object provided to this object.

All scripts can see oExam, oForm, and oCandidate.

If the parent object is a section object, then the script will also see oCurSection.

If the parent object is a presentation object, then the script will also see oCurSection and oCurPresentation.

If the parent object is an item then the script will also see oCurSection, oCurPresentation, and oCurItem.

Public Attributes:

oParent : Object

The parent UTD object. It could be a form, a section or a presentation.

Public Operations:

Execute (sScript : BSTR) :

Executes the script code passed in the sScript parameter.

EvaluateBool (sScript : BSTR) : Boolean

Evaluates the expression parsed as the sScript parameter. Returns TRUE or FALSE. Raises an error if the code is not an expression returning a boolean.

EvaluateVar (sScript : BSTR) : Variant

Evaluates the expression passed in the sScript parametert. Returns a variant depending on the return type of the script. Raises an error if the code is not an expression.

SetScope (oSection : ISection *, oPresentation : IPresentation *, oItem : IItem *) :

Set the parent section, presentation, and item. The script will have access to these as the oCurSection, oCurPresentation, and oCurItem objects. Any/All may be passed as NULL parameters indicating that the script does not have access to that particular object.

AddObject (sName : BSTR, pObject : IDispatch *) : HRESULT

Add an object into the global script scope.

Protected Operations:

LoadGlobalScript (oStream : POLESS.IcStorage) :

Called by the test driver to initialize the global script that is common to all later scripts. This will be shared by all instances of cScript.
The oStorage parameter is a storage containing an attribute set and a stream containing the global script.

UTDCore.IUnitTimer (Interfaces)

A plugin to perform timing across testing units.

Public Attributes:

lSecondsElapsed : Long

The seconds elapsed.

lSecondsRemaining : Long

The seconds remaining. If the unit is not timed, this will return -1.

lSecondsTotal : Long

The total seconds available for the unit.

datElapsed : DATE

The time elapsed. Same as lSecondsElapsed.

datRemaining : DATE

The time remaining. If the unit is not timed, this will return -1. Same as lSecondsRemaining

datTotal : DATE

The total time available for the unit. Same as lSecondsTotal.

Public Operations:

PresentationActive (oPresentation : UTDCore.IPresentation) :

This is called by the test driver to inform the timer that the presentation is completely active. All plugins have loaded their data, displayed and are operational.

PresentationInactive () :

This is called by the test driver to inform the timer that the presentation is about to become inactive. The user has selected a direction to go from the item. The Item and the navigation has cleared it for movement.

After this call the plugins will be hidden and unloaded.

AddTime (nMinutes : Integer) :

Adds time to the current form or section. Time is specified in minutes.

Starting (oChildren : UTDCore.IEvents, oContainerNotifyHelm : UTDCore.IContainerNotifyHelm) :

Informs the plugin that the section or form is starting. If the plugin is defined at the form level then this is called when the form starts. If the plugin is defined at the section level then this is called when the section starts.

It gives it the collection of children to time across. This collection can include cSection objects and cPresentation objects.

Ending () :

Informs the plugin that the section or form is ending. If the plugin is defined at the form level then this is called when the form ends. If the plugin is defined at the section level then this is called when the section ends.

Pause () :

Pauses the timer

Resume () :

Resumes the timer. (After a Pause())

UTDCore.INavigate (Interfaces)

Interface supportted by plugins that a handle section navigation.

Implementer of iNavigate must also implement a iContainerNotifyHelm

Derived from UTDCore.IContainerNotifyHelm

Public Attributes:

nPresentations : Integer

The number of counted presented to be presented within the scope of the navigation. Each item in a compound items count as a one.

bItemsDetermined : Boolean

TRUE if the number of items to present within the scope of the navigation can be determined. Adaptive items or sections can prevent this.

Public Operations:

bNavigateAllowed (eDirect : UTDCore.eDirection) : Boolean

Returns TRUE if navigation in the direction supplied sis allowed. It is the helm's job to query this before every presentation.

This is a combination of the resource file plugin data and the current state of the section.

Example implementation:
The test designer is allowed to configure if the previsous button appears. But if we are on the first presentation of a section it will not appear even if configured.

Starting (oSection : UTDCore.ISection) :

The section is starting.

Ending () :

The section is ending.

UTDCore.IItems (Interfaces)

Collection of UTDCore.IcItem. Note this is not the UTDCore.IItem interface implemented by an item plug-in.

Derived from UTDCore.Collection

UTDCore.ICategory (Interfaces)

A category of sections or items. For example: "Reporting", "Objectivies", "Scoring"

Public Attributes:

sName : String

The name of the category.

bComplete : Boolean

If TRUE the category must contain all sections or items defined at a level.

bDuplicate : Boolean

If TRUE the category allows sections or items defined more than once at a level.

sDescription : String

The description of the category.

colMembers : UTDCore.IEvents

The collection of sections or items that apply to this category .

colSubCategories : UTDCore.ICategories

Collection of UTDCore.cCategory. (AKA sub-categories).

eContents : eCategoryContents

Read-only. The allowable types of members in the colMembers.

```
enum {
  eCategoryContent_Anything = 0,
  eCategoryContent_Sections = 1,
  eCategoryContent_Items = 2,
  eCategoryContent_Categories = 3,
} eCategoryContents;
```

oScoring : UTDCore.IScore

The scoring plugin for the form.

UTDCore.IPersistInstanceStore (Interfaces)

Interface for a plugin that wants to be persistant in the instance file as a storage.

Public Attributes:

IsDirty : Boolean

Returns TRUE if the object needs to save state to the instance file.

Public Operations:

Save (oStorage : POLESS.iStorageVB) :

Called by the test driver to save the plugin data to the instance file.

Reload (oStorage : POLESS.iStorageVB) :

Called by the test driver to reload the plugin data from the instance file. The IPersistResource*::Load will be called next.

UTDCore.IcResults (Interfaces)

A collection of Events will contain an IcResults interface for every result. It is a container for the IResults plug-in. This is not the IResults interface that the plug-in implements.

Derived from cEvent

Public Attributes:

oResults : UTDCore.IResults

The plugin that writes the results.

UTDCore.IcItem (Interfaces)

A collection of Items or Events will contain an IcItem interface for every item. This interface provides general item information including attributes specified in the <item> tag of the XXL. This is not the UTDCore.IItem interface that the plug-in implements.

It is possible to QueryInterface this IcItem for the IItem interface. This allows a report, results, or scoring plug-in to call methods on the IItem interface.

Derived from UTDCore.IItem

Public Attributes:

bPresented : Boolean = False

True if item has been presented to the candidate. Set by item component.

sName : String

Read-only item name.

sTitle : String

Presentation title. Read-only

fWeight : Double = 1

The item weigthing relative to other items. Normally 1. If multiple weigths per item are required they can be attached in the custom attributes. The weigth can be negative.

colCustomAttributes : UTDCore.IAttributes

Collection of custom attributes. Read-only.

lSecondsElapsed : Long

The seconds spent on the item.

bScored : Boolean = TRUE

Read-only. TRUE if the item is to be scored.
If false fWeight and fScoreCandidateWeighted will all be 0.

bSkipAllowed : Boolean

Read-only property. If TRUE the item can be skipped. If TRUE the bProceed() method on iItem interface will not be called.

fScoreCandidateWeighted : Double

The weighted value of the candidate for the item. The judged value for the item will by multiplied by the weight for this value.

sPluginName : String

Read-only plugin name. The prog ID of the plugin.

oSection : UTDCore.ISection

Read-only property of the section the item was deliveried in.

oPresentation : UTDCore.IPresentation

Read-only property of the presentation the item was deliveried in.

datElapsed : DATE

The time spent on the item. Same as lSecondsElapsed, but sent as a COM DATE type.

UTDCore.IScore (Interfaces)

Interface supportted by plugins that a handle scoring of sections or exams.

Public Attributes:

eStatus : UTDCore.eScoreStatus

The eScoreStatus as follows.

fReportedScoreCut : Double = -1

The cut score to report to the candidate and results. Usually scaled. If this does not apply to the plugin return -1.

fReportedScoreMax : Double = -1

The maximum score to report to the candidate and results. Usually scaled. If this does not apply to the plugin return -1.

fReportedScoreCandidate : Double = -1

The achivied score to report to the candidate. Usually scaled. If this does not apply to the plugin return -1.

oEvents : IEvents *

Tthe collection of children to score. If the plugin is on a section, then this collection can include ISection and IPresentation objects. If the plugin is on a category, then this collection can include ISection, and IItem objects.

Public Operations:

bStop () : Boolean

Test Driver will call this method to see if delivery should be stopped from current delivery unit (section or form).

For example an adaptive exam may return false when it is clear if the candidate has master or not mastered the material.

GetScoreDisplay () : String

Returns the score in a form to be displayed on the screen.

GetScoreResults () : Byte()

Returns the score in a form to be written to the results file. The score is in the form of a safe array of bytes.

UTDCore.IExam (Interfaces)

The exam instance. The root object of the driver exposed classes.

Derived from cContainerFrameT

Public Attributes:

sName : String

Read-only exam name.

sLanguage : String

The language of the exam content. Read-only.
(UTDCore 2.0.0.47 - This currently returns a blank string)

sVersion : String

The version of the exam. Read-only.

oActiveForm : UTDCore.IForm

The current form. Only one form can be active for an exam.

eStatus : UTDCore.eExamStatus

The eExamStatus as follows:

eExam_Initializing
eExam_Starting
eExam_InProgress

eExam_Ended

eExam_Terminating

bShowResponseComment : Boolean

If TRUE the candidate should not see their response during the comment period.

colCustomAttributes : UTDCore.IAttributes

Collection of custom attributes. Read-only.

sTitle : String

Exam title. Read-only

colCategories : UTDCore.ICategories

The collection of all categories.

oResourceRoot : Object

The read-only reference to the root storage of the POLESS resource file. Interfaces: POLESS.cStorageRoot, iStorage, iRootStorage, iStorageVB.

oInstanceRoot : Object

The read-only reference to the root storage of the POLESS exam instance file. Interfaces: POLESS.cStorageRoot, IStorage, IRootStorage, IStorageVB.

oRegistry : UTDCore.IRegistry

Returns the single instance of the registry object. Read-only.

Public Operations:

Minimize () :

Minimumizes the main windows of UTD. Can be called be simulations to get UTD out of the way.

Restore () :

Restore the main windows of UTD. Can be called be simulations to restore UTDto the screen.

GetScript () : UTDCore.IScript

Returns a script object to be used by a plug-in.

SaveInstance () :

Save the state of the test driver to the instance file. All active objects are asked to serialize themselves to the instance file.

Any one with a reference to this object can initiate this process. It could be an item plugin that wants to save user interaction and state information.

The driver will save between each presentation. It may also save at various other events or time intervals.

GetTransfer () : UAS.ITransfer

Returns a UAS Transfer object to transfer results or other files to the data center.
The IExam object does not keep a reference to this object.

GetPrint () : IUASPrint

Returns a UAS print object to save for reprint score and other reports.
The IExam object does not keep a reference to this object.

FatalError (lNumber : Long, sSource : String, sDescription : String) :

Reports an error in a plugin to the test driver.

Normally errors will be reported back to the test driver via an HRESULT and supportting IErrorInfo. If the error was generated from something other than test driver initiated call, the plugin should report the error via this method.

An example usage could be the user clicks a button on your plugin item. The resulting actions generate an error.

Warn () :

Writes a 'Warning' level trace messate to the instance file.

Info () :

Writes a 'Information' level trace messate to the instance file.

Quit () :

Tells the exam that the candidate is quitting the exam. Normally called by the Navigation plugin in response to the eDirection_Quit. The navigation plugin should confirm this decision with the candidate.

This causes the test driver to stop delivery and shutdown execution cleanly. The administration system is informed that the candidate quit the exam.

GetMsgBox () : UTDCore.IMsgBox

Returns a message box object. Any plug-ins that wish to display a message box should use this object whenever possible. It provides language independence, and a consistent look and feel.

StartExam () :

Tells the exam that the candidate has accept the exam and started. Normally called by the Navigation plugin in response to the eDirection_Start.

The administration system is informed that the exam has started and the candidate has accepted it.

UTDCore.IRegistry (Interfaces)

Alows registry look up for translating strings

For examples:

<base href="%REG:DSAGRAPGHICS">

into

<base href="V:\DSA\">

Looks at [HKEY_LOCAL_MACHINE\SOFTWARE\Prometric\UTD\Keywords]

Public Operations:

Lookup (sKey : String) : String

Retrieves the string by finding the key in the registry and returning the value.

Translate (sInString : String) : String

Translates each occurance of %REG:x% into the equivilent of "x" in the registry.

UTDCore.IAttribute (Interfaces)

Public Attributes:

sName : BSTR

Get the attribute name string.

value : VARIANT

Get the attribute value.

UTDCore.IHelm (Interfaces)

Interface implemented by plug-ins that a handle navigation control or review. Like Next, Previous or Incomplete. Only User interface. Talks to the iNavigate plug in to perform the actual navigation.

Public Attributes:

oSection : UTDCore.ISection

The section this helm is in.

Public Operations:

PresentationStarting (oPresentation : UTDCore.IPresentation, oContainerNotifyHelm : UTDCore.IContainerNotifyHelm, oContainerNotify : UTDCore.IContainerNotify, bComment : Boolean)

Called by the test driver to inform the plugin that a new presentation is starting. The plug-in may hold the IPresentation, IContainerNotify, and IContainerNotifyHelm objects until the PresentationEnding. See IContainerNotify::Activate. The IContainerNotifyHelm interface as a way to notify the test driver of movement.

PresentationEnding :

Called by the test driver to inform the plug-in the presentation is ending. After calling this method, the driver expects the plug-in will release the IPresentation, IContainerNotify, and IContainerNotifyHelm objects passed in PresentationStarting.

UTDCore.IcReport (Interfaces)

A collection of Events will contain an IcReport interface for every report. It is a container for the IReport plug-in. This is not the IReport interface that the plug-in implements.

Derived from cEvent

Public Attributes:

oScoreReport : UTDCore.IReport

The score report plugin.

UTDCore.eDirection (Interfaces)

Enumeration of the possible directions to proceed from a presentation.

eDirection_Next
eDirection_Previous
eDirection_TimeoutImmedidate (now do not wait)
eDirection_TimeoutAfter (after current presentation)
eDirection_Review (Goto the review presentation)
eDirection_Jump (Goto a specific presentation)

eDirection_Flag
eDirection_Help
eDirection_FirstIncomplete
eDirection_FirstSkipped
eDirection_FirstPresentation
eDirection_FirstMarked
eDirection_End
eDirection_Comment
eDirection_Start
eDirection_Quit

UTDCore.IPersistInstanceStream (Interfaces)

Interface for a plugin that wants to be persistant in the instance file as a property stream.

Public Attributes:

IsDirty : Boolean

Returns TRUE if the object needs to save state to the instance file.

Public Operations:

Save (oStream : POLESS.IStreamVB) :

Called by the test driver to save the plugin data to the instance file.

Reload (oStream : POLESS.IStreamVB) :

Called by the test driver to reload the plugin data from the instance file. The IPersistResource*::Load will be called next.

UTDCore.ePluginModes (Interfaces)

Enumeration of the possible modes that an active document

ePluginMode_Author (Not supported)

ePluginMode_Compile
ePluginMode_Delivery
ePluginMode_Rescore (Not suportted)
ePluginMode_ProcessResults (Not suportted)

UTDCore.IPersistResourceStream (Interfaces)

Interface for a plugin that wants to be persistant in the resource file as a stream.

Public Operations:

Save (oStream : POLESS.IStreamVB) :

Called by the compiler to save the plugin data to the resource file.

Load (oStream : POLESS.IStreamVB) :

Called by the test driver to load the plugin data from the resource file.

UTDCore.ISelection (Interfaces)

Selects items, sections or forms from a category.

Public Attributes:

bMore : Boolean

Returns TRUE if there are more objects to get. Read-only.

nCountTotal : Integer

Returns the total count of objects (items, sections or forms) that it has in its pool to select from. It will return -1 if this is not quantifiable.

nCountSelected : Integer

Returns the selected count of objects (items, sections or forms) that it will return from the pool. It will return -1 if this is not quantifiable. Like items in an adaptive section.

oGroup : UTDCore.ISection

Read-write.

The group that contains this selection plugin. This is not set for a form selection plugin.

oEvents : UTDCore.IEvents

Read-write.

The events collection to select from.

Public Operations:

Reset () :

This gives the selection plug a chance to prepare the data for the first selection. Some selection plugins like "random/exshastive" make pre-determine all the items for selection a store them in its internal memory and the instance file.

GetNext () : Object

Gets the next item, section or form. If there are no more, them NULL AKA Nothing is returned.

UTDCore.IGroups (Interfaces)

Collection of UTDCore.IGroup

Derived from UTDCore.Collection

UTDCore.Collection (Interfaces)

Base UTD collection. All other collection classes follow this design.

Public Attributes:

Count : Long

Returns the number of members in a collection.

_NewEnum : IUnknown * *

Returns a Visual Basic enumerator object.

Public Operations:

Item (Index : Variant) : Variant

Returns a specific member of a Collection object either by position or key.

UTDCore.eScoreStatus (Interfaces)

The scoring status for a form/section/group/category, determined by the scoring plugin.

eScore_NonGraded
eScore_Failure
eScore_Pass

UTDCore.eStates (Interfaces)

The states of the test driver.

eStateStart -- Initializing the test driver & reading the resource file.
eStateSectionDelivery -- Delivering a section.
eStateSectionComment -- Commenting all sections.
eStateSectionFeedBack -- Feedback on all sections.
eStateScoreReporting -- Rendering and printing a score report.
eStateResults -- Writing results.
eStateFinish -- Closing down the test driver.

UTDCore.IPersistResourceStore (Interfaces)

Interface for a plugin that wants to be persistant in the resource file as a storage.

Public Operations:

Save (oStorage : POLESS.iStorageVB) :

Called by the compiler to save the plugin data to the resource file.

Load (oStorage : POLESS.IStorageVB) :

Called by the test driver to load the plugin data from the resource file.

ValidateResource (oStorage : POLESS.iStorageVB, oCompilerServices : UTDCore.ICompilerServices) :

Validates the compiled resource for the plugin. The plugin must raise an automation error that describes any problems with the source.

The combination of the amalogmated storage for the XXL and the referenced files should resolve all references. For example all HREFs to HTML and JPEGs should resolve.

UTDCore.eExamStatus (Interfaces)

Enumeration of possible exam delivery states.

eExam_Initializing (loading resource file)
eExam_Starting (delivering, candidate not yet "started")
eExam_InProgress (delivering, candidate officially "started")
eExam_Ended (delivering, candidate officially "ended")
eExam_Terminating (cleaning up from memory)

UTDCore.IItem (Interfaces)

Interface supportted by plugins that a handle items and simulators.

Public Attributes:

oSection : UTDCore.ISection

The section we are in.

fScoreMinimum : Double = 0

The lowest possible score. Must be less than or équal to the nominal score. It can be negative.

fScoreNominal : Double = 0

The score if the candidate takes no actions. Must be less than or equal to the maximum score. Must be greater or equal to the minimum score. Normally zero.

fScoreMaximum : Double = 1

The highest possible score. Must be greater than or equal to the nominal score.

fScoreCandidate : Double = 0

The score received by the candidate. Requests that the plugin judges the item.

bComplete : Boolean = False

True if Item is complete.

bSkipped : Boolean = False

True if item was skipped by the candidate.

Public Operations:

bProceed (eDirect : UTDCore.eDirection) : Boolean

Test Driver will call this method to see if it is OK to move off the current presentation. The parameter indicates the direction.

GetCorrectAnswerDisplay () : String

Returns the correct answer in a form to be displayed on the screen.

GetCorrectAnswerResults () : Byte()

Returns the correct answer in a form to be written to the results file. The correct answer is in the form of a safe array of bytes.

GetResponseDisplay () : String

Returns the candidate response in a form to be displayed on the screen.

GetResponseResults () : Byte()

Returns the candidate response in a form to be written to the results file. The response is in the form of a safe array of bytes.

PresentationStarting (oPresentation : UTDCore.IPresentation, oContainerNotify : UTDCore.IContainerNotify, bReadOnly : Boolean) :

Called by the test driver to inform the plugin that a new presentation is starting.

PresentationEnding () :

Called by the test driver to inform the plugin thae presentation is ending.

UTDCore.IPlugin (Interfaces)

This Interface will be supportted by all UTD plugins.

All plugins must implement the following interfaces:
iPlugin

All plugin must implement one of the following to persist into the resource file at compile time:
IPersistResourceSet, IPersistResourceStore or IPersistResourceStream

All plugin may implement one of the following to persist into the instance file during delivery time:
IPersistInstanceSet, IPersistInstanceStore or IPersistInstanceStream

if you are visible one of these: iHelm, iItem or iDisplay. It can be an iHelm and an iItem.
If you wish to be contained as an active document you must support the following: iDataObject, IoleInPlaceObject, IoleInPlaceActiveObject , IoleDocument, IOleDocumentView and IoleCommandTarget.

or one of these if you are invisible:
iScore, iReport, iResults, iNavigate, iTimer.

iNavigate must also implement a iContainerNotifyHelm

Public Attributes:

eMode : ePluginModes

The mode the plugin should operate in.

Set by the consumer of the plugin (the driver or compiler).

Public Operations:

ValidateSource (oSource : POLESS.IStreamVB, oCompilerServices : UTDCore.ICompilerServices) :

Validates the source XXL for the plugin. The plugin must raise an automation error that describes any problems with the source. The source is not required to be complete only the portions provided should be verified.

If the contents of the stream is Unicode it will be marked with the BOM (byte order mark) as defined be unicode standard (www.unicode.org). The BOM is normally FFFE.

If the stream contains ASCII or UTF-8, no BOM will be included.

The oCompilerServices interface is provided to offer additional features and information to the plugins.

Unload () :

Unload data and references to UTD objects.

Load (oExam : UTDCore.IExam, oCandidate : iAttendance) :

Load with references to UTD objects. Only called during exam delivery.

UTDCore.IContainerNotifyHelm (Interfaces)

This interface consumed by plugins to inform the container to navigate.

Public Operations:

RequestMove (eDirect : UTDCore.eDirection, sPresentation : String) :

Requests that the driver proceed in the direction specified. The driver next requests this movement from the navigation plugin.

The second parameter optional specifies the presentation. This is only used for the JUMP.

UTDCore.IReport (Interfaces)

Interface supported by plugins that handle printing of the score reports and other other material like stylus ("hand-outs").

Public Operations:

PrintReport () :

Prints the score report

UTDCore.IForm (Interfaces)

A exam form.

Derived from UTDCore.IContainerNotifyHelm

Public Attributes:

colChildren : UTDCore.IEvents

The collection of all top level sections of the exam. Read-only.

colCustomAttributes : UTDCore.IAttributes

Collection of custom attributes. Read-only.

sName : String

Read-only form name.

sTitle : String

Form title. Read-only

colItemsChosen : UTDCore.IItems

The collection of all items chosen (that is, returned by a selection plugin) in the exam. This is regardless of their section.

colAllSections : UTDCore.ISections

The collection of all sections of the exam regardless of their level.

datStart : Date

The form start date and time.

datFinish : Date

The form finish date and time.

oTimer : UTDCore.IUnitTimer

The timer for the form.

oScoring : UTDCore.IScore

The scoring plugin for the form.

sVersion : String

The version of the form.

colDelivered : UTDCore.IEvents

The collection of all delivered top level sections of the form. Read-only.

nCurIndex : Long = 0

Index of last delivered event in colDelivered. Read-only.

eStatus : UTDCore.eScoreStatus

Returns the value of oForm.oScore.eStatus.

colAllPresentations : UTDCore.IPresentations

The collection of all presentations of the exam regardless of their level.

colPresentationsChosen : UTDCore.IPresentations

The collection of all presentations chosen (that is, returned by a selection plugin) in the exam. This is regardless of their section or group.

colSectionsChosen : UTDCore.ISections

The collection of all sections chosen (that is, returned by a selection plugin) in the exam. This is regardless of their level.

colAllItems : UTDCore.IItems

The collection of all items of the exam regardless of their level.

colAllGroups : UTDCore.IGroups

The collection of all groups of the exam regardless of their level.

colGroupsChosen : UTDCore.IGroups

The collection of all groups chosen (that is, the selection plugin for this group has been Reset()) in the exam. This is regardless of their level.

Public Operations:

ChildNext () :

Requests that the driver proceeds to the next event on the form. This can be a section, a result, or a score report.

ChildPrevious () :

Requests that the driver proceeds to the previous event on the form. This can be a section, a result, or a score report.

ChildGoto (vSection : Variant) :

Requests that the driver proceeds to a named or numbered event on the form. This can be a section, a result, or a score report.

datMinRemaining () :

Minimum time remaining for this exam.

UTDCore.IAttributes (Interfaces)

Collection of UTDCore.IAttribute

Derived from UTDCore.Collection

UTDCore.IPresentations (Interfaces)

Collection of UTDCore.cPresentation

Derived from UTDCore.Collection

UTDCore.ISections (Interfaces)

Collection of UTDCore.cSection

Derived from UTDCore.Collection

UTDCore.IDisplay (Interfaces)

Interface supported by plugins that a handle title bars, displays, non-answered items and summaries.

Public Attributes:

oSection : UTDCore.ISection

The section this display is in.

Public Operations:

PresentationStarting (oPresentation : UTDCore.IPresentation, oContainerNotify : UTDCore.IContainerNotify) :

Called by the test driver to inform the plugin that a new presentation is starting. The plug-in may hold the IPresentation and IContainerNotify objects until the PresentationEnding. See IContainerNotify::Activate.

PresentationEnding () :

Called by the test driver to inform the plug-in the presentation is ending. After calling this method, the driver expects the plug-in will release the IPresentation and IContainerNotify objects passed in PresentationStarting.

UTDCore.ISection (Interfaces)

Represents one exam sections or sub-section. May contain presentations or sections.

Derived from cEvent

Public Attributes:

oNavigation : UTDCore.INavigate

The navigation plugin for this section.

colCustomAttributes : UTDCore.IAttributes

Collection of custom attributes. Read-only.

sName : String

Read-only section name.

sTitle : String

Section title. Read-only.

oScoring : UTDCore.IScore

The scoring plugin for the section.

oScriptStart : UTDCore.IScript

The script to execute before this object executes.

oScriptFinish : UTDCore.IScript

The script to execute after this object executes.

oScriptConditional : UTDCore.IScript

This expression script is evaluated. If TRUE this object is deliveried.

oReview : UTDCore.IPresentation

The review presentation for the section. It can be nothing if the section does not have a review defined for it.

oCurChild : Object

The current child being deliveried. This can be a cSection or a cPresentation.

datStart : Date

The section start date and time.

datFinish : Date

The section finish date and time.

colChildren : UTDCore.IEvents

The collection of presentations or sub-sections for the section.

oSelection : UTDCore.ISelection

The selection plugin for this section.

oTemplate : UTDCore.cTemplate

The tmeplate for the section.

oTimer : UTDCore.iTimer

The timer plugin for the section.

dtTimeRemaining : DATE

Returns the time remaining in the section. It get the seconds remaining from the iTimer plugin. It then converts it to a variant DATE type (aka VT_DATE).

colDelivered : UTDCore.IEvents

The collection of presentations or sub-sections for the section that have been presented. They are ordered by the order they appeared in.

bComment : Boolean = FALSE

The commenting mode of the section. Normally set by the Navigation.

bItemsReadOnly : Boolean = FALSE

The read-only mode of the items in the section. Normally set by the Navigation. Read only items cannot have their responses changed.

nCurChild : long = 0

Index of last delivered event in colDelivered. Read-only.

Public Operations:

ChildNext () :

Requests that the driver proceeds to the next presentation or child section.

ChildPrevious () :

Requests that the driver proceeds to the previous presentation or child section.

ChildGoto (vPresentation : Variant) :

Requests that the driver proceeds to a named or numbered presentation or child section..

UTDCore.IResults (Interfaces)

Interface supported by plugins that handles writing of the candidate results.

Public Operations:

WriteResults () :

Write the exam results.

UTDCore.eCategoryContent (Interfaces)

Enumeration of possible values for ICategory::eContents

// These types of categories that correspond to the XXL categorycontents
eCategoryContent_Anything
eCategoryContent_Sections
eCategoryContent_Items
eCategoryContent_Categories

UTDCore.IEvents (Interfaces)

IEvents contains deliverable classes as IDispatch objects. They must be queried individually to determine their exact type. Which types of events are returned depends on where the collection comes from.

IEvents can contain:
IExam, IForm, ISection, IcItem, IPresentation, IcItem, IcReport, and IcResults.

Derived from UTDCore.Collection

UTDCore.IPersistInstanceSet (Interfaces)

Interface for a plugin that wants to be persistant in the instance file as a property set.

Public Attributes:

IsDirty : Boolean

Returns TRUE if the object needs to save state to the instance file.

Public Operations:

Save (oPropSet : POLESS.IPropertyStorageVB) :

Called by the test driver to save the plugin data to the instance file.

Reload (oPropSet : POLESS.IPropertyStorageVB) :

Called by the test driver to reload the plugin data from the instance file. The IPersistResource*::Load will be called next.

Administration System

This is one or more components that make up the administration system. This will vary by channel and time.

UAS.ITransfer (Interfaces)

Administration interface to transfer results and other files back to the data center. It includes routing.

It is also emulated by UTD2ETS and Launchtest components that emulate the UAS.

UAS.IPrint (Interfaces)

The administration system interface for save score reports for reprint. Also handles initial printing.

It is also emulated by UTD2ETS and Launchtest components that emulate the UAS.

UAS.IAppointment (Interfaces)

This interface is part of the Unified Administration System. It allows access to the candidate information for the candidate taking this exam.

It is also emulated by UTD2ETS and Launchtest components that emulate the UAS.

APPENDIX B-
POLESS CLASSES AND INTERFACES

TABLE OF CONTENTS

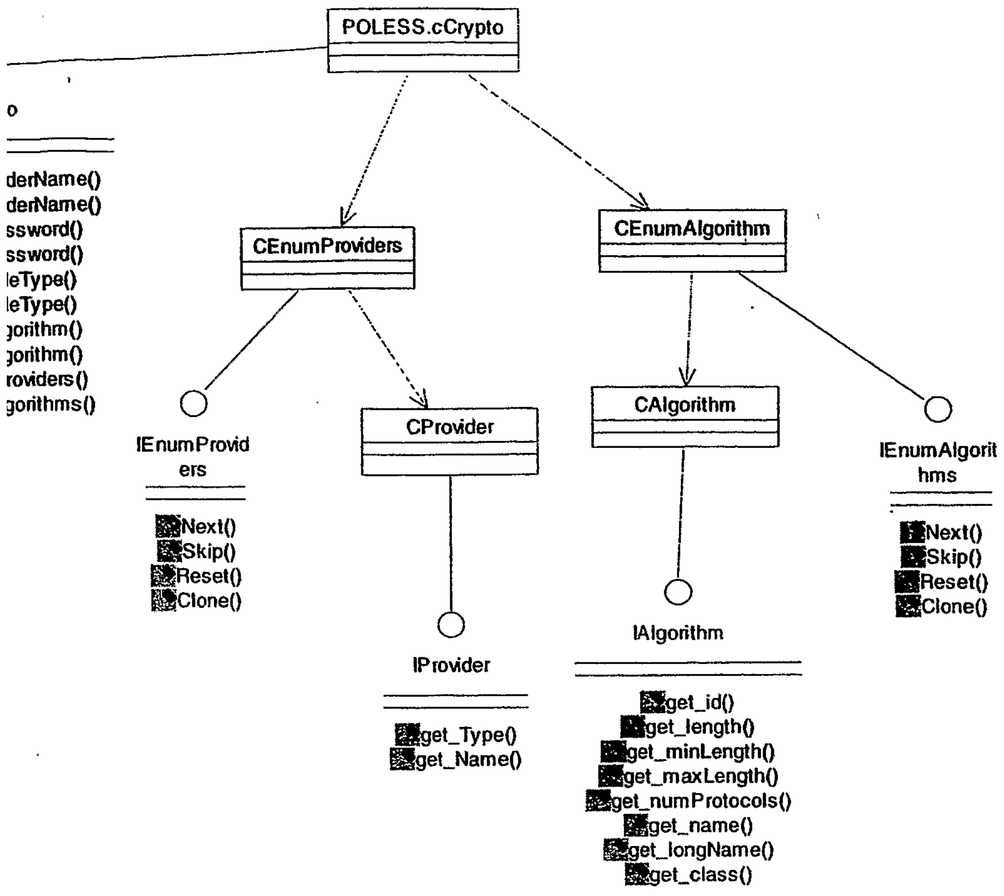
TABLE OF CONTENTS 2

SELECTED LOGICAL VIEW REPORT 3

LOGICAL VIEW 3
CAlgorithm *3*
CEnumAlgorithm *3*
CEnumProviders *3*
CProvider *3*
IAlgorithm *4*
ICrypto *4*
IEnumAlgorithms *5*
IEnumProviders *5*
IFileRoot *7*
POLESS.IPropertyStorageAmalgamated *8*
POLESS.IPropertyStorageVB *8*
POLESS.IStorageAmalgamated *9*
POLESS.IStorageVB *10*
POLESS.IStreamVB *12*
POLESS.IcStorage *13*
POLESS.IcStorageRoot *13*
POLESS.cCrypto *14*
POLESS.cFileRoot *14*
POLESS.cPropertyStorage *15*
POLESS.cPropertyStorageAmalgamated *15*
POLESS.cStorage *15*
POLESS.cStorageAmalgamated *15*
POLESS.cStorageRoot *15*
POLESS.cStream *15*
POLESS.iStorageVB *15*
POLESS.iStreamVB *15*

TOTALS: 16

LOGICAL PACKAGE STRUCTURE 16



SELECTED LOGICAL VIEW REPORT

Logical View

CAlgorithm

CEnumAlgorithm

CEnumProviders

CProvider

IAlgorithm

Public Operations:

get_id (pAlgId : LONG*) : HRESULT

get_length (pdwLength : LONG*) : HRESULT

get_minLength (dwMinLength : LONG*) : HRESULT

get_maxLength (dwMaxLength : LONG*) : HRESULT

get_numProtocols (dwNumProtocols : LONG*) : HRESULT

get_name (sName : BSTR*) : HRESULT

get_longName (sName : BSTR*) : HRESULT

get_class (peAlgClass : eAlgorithmClass) : HRESULT

ICrypto

Encryption/Decryption interface.

Public Operations:

get_ProviderName (psProviderName : BSTR*) : HRESULT

Returns the name of the Cyrto provider.

put_ProviderName (sProviderName : BSTR) : HRESULT

Sets the name of the Cyrto provider.

get_Password (sPassword : BSTR* = "") : HRESULT

Used for Sponsor resource files only.

put_Password (sPassword : BSTR) : HRESULT

Used for Sponsor resource files only.

get_FileType (eFileType : eFILE_TYPE) : HRESULT

put_FileType (eFileType : eFILE_TYPE) : HRESULT

get_Algorithm (eAlgorithm : eALGORITHM) : HRESULT

put_Algorithm (eAlgorithm : eALGORITHM) : HRESULT

EnumProviders (eProvType : eProviderType, ppenum : IEnumProviders) : HRESULT**

EnumAlgorithms (sProvName : BSTR, eAlgClass : eAlgorithmClass, ppenum : IEnumAlgorithms) : HRESULT**

IEnumAlgorithms

Public Operations:

Next (ppProv : IAlgorithm) : HRESULT**

Returns the next algorithm interface, or NULL if there are no more algorithms.

Skip (celt : ULONG) : HRESULT

Skips over the next specified number of algorithms.

Reset () : HRESULT

Resets the enumerator to the beginning.

Clone (ppenum : IEnumAlgorithms) : HRESULT

Creates another enumerator that contains the same enumeration state as the current one.

IEnumProviders

Public Operations:

Next (ppProv : IProvider) : HRESULT**

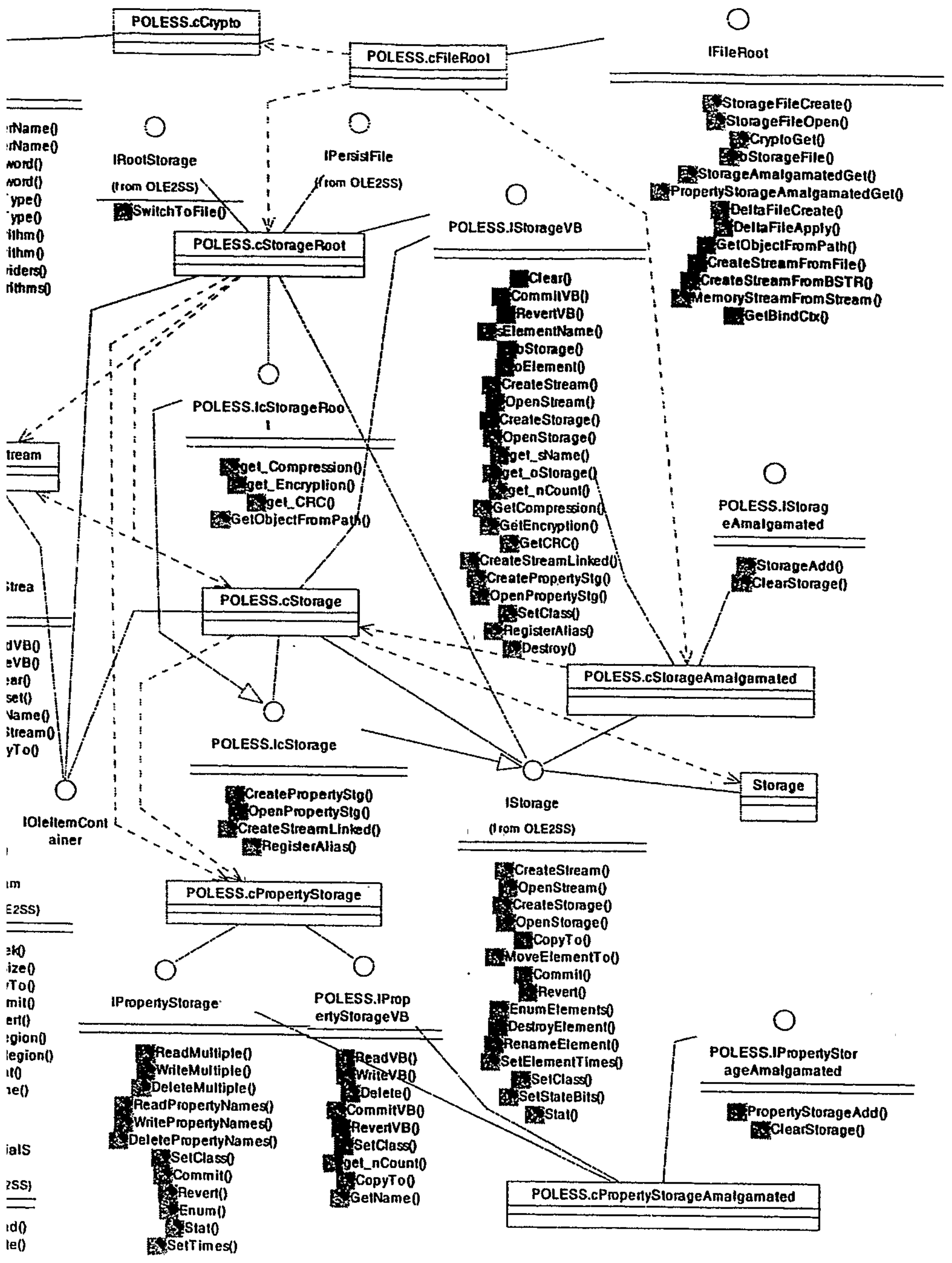
Returns the next provider interface, or NULL if there are no more providers.

Skip (celt : ULONG) : HRESULT

Skips over the next specified number of providers.

Reset () : HRESULT

Resets the enumerator to the beginning.



Clone (ppenum : IEnumProvider) : HRESULT**

Creates another enumerator that contains the same enumeration state as the current one.

IFileRoot

The root POLESS interface. This interface should only be used to open one file ata a time, and should not be released until all other storage, stream and propertystorage interfaces and released and the file is ready to be closed.

Public Operations:

StorageFileCreate (sFileName : String, eBlockSize : eBLOCK_SIZE, eAccessMode : eACCESS_MODE, bCompression : Boolean, bEncrypt : Boolean, oCrypto : POLESS.cCrypto, bCRC : Boolean) : iStorage

Creates a new storage file. Returns the root storage interface. Marks the new structured storage file as a POLESS file by storing the CLSID of this class in a stream in the root storage.

StorageFileOpen (sFileName : String, eAccessMode : eACCESS_MODE) : iStorage

Open an existing storage file. Returns the root storage interface.

The CRC is checked if enabled and read access mode is selected. An error is returned if the CRC fails to match.

CryptoGet () : POLESS.cCrypto

Gets a default configured crypto class. It should be set and used on the open or create of of the storage file.

bStorageFile (sFileName : String) : Boolean

Return TRUE if the file provided is an OLE structured sotrage file and a POLELESS storage file.

StorageAmalgamatedGet () : POLESS.cStorageAmalgamated

Gets an empty cStorageAmalgamted.

PropertyStorageAmalgamatedGet () : POLESS.cPropertyStorageAmalgamated

DeltaFileCreate (sFileNameOriginal : String, sFileNameUpdate : String, sFileNameDelta : String, bEncrypt : Boolean, oCrypto : POLESS.cCrypto, bCRC : Boolean) :

Create a POLESS difference file. It compares the original poless file to the updated poless file and create a delta poless file.

The delta file contains
branch additions and branch deletions to the orginal poless tress to create the updated poless tree.
it contains the CRC of the original file and
the CRC of the update file.

DeltaFileApply (sFileNameOriginal : String, sFileNameUpdate : String, sFileNameDelta : String) :

Applies a POLESS delta file. It applies to the original poless file to the delta poless file and create an updated poless file.

The CRC in the delta file for the original poless file in compared to the original file's calculated CRC.
If they match then the deltas are allied to create the update
poless file. The CRC of the update file is calculated and compared to the update file CRC in the delta file.

GetObjectFromPath (sFullPath : BSTR, eAccessMode : eACCESS_MODE, ppDisp : IDispatch **) : HRESULT

Uses monikers to retrieve the object named by the path. Returns a IDispatch pointer to the object retrieved.

CreateStreamFromFile (sName : BSTR, ppDisp : IDispatch) : HRESULT**

Creates a structured storage stream and populates it with the contents of the file.

CreateStreamFromBSTR (sIn : BSTR, ppDisp : IDispatch) : HRESULT**

Creates a structured storage stream and fills it with the specified BSTR.

MemoryStreamFromStream (pStreamIn : IStream*, ppDisp : IDispatch) : HRESULT**

Used to copy a stream to a newly created memory stream object, seek pointers for both streams are reset to beginning of stream after operation.

GetBindCtx (ppBindCtx : IBindCtx) : HRESULT**

Returns the static bind context that is used for creating monikers.

POLESS.IPropertyStorageAmalgamated

Public Operations:

PropertyStorageAdd (oPropertySet : IDispatch*, bEnd : VARIANT_BOOL = TRUE) : HRESULT

Add a PropertySet to the collection of PropertySets.

ClearStorage () : HRESULT

Clears the collection of PropertySets.

POLESS.IPropertyStorageVB

Manages the persistent properties of a single property set.

Public Operations:

ReadVB (sName : BSTR, ppvVal : VARIANT) : HRESULT**

Read the value of a specified property from the property set.

WriteVB (sName : BSTR, pvVal : VARIANT*) : HRESULT

Write a value for a specified property to the property set. If the property does not exist the property/value pair will be created. If the property already exist the value will be updated if open in eAccess_Write mode.

Delete (sName : BSTR) : HRESULT

Remove a property from the property set.

CommitVB (grfFlags : DWORD) : HRESULT

RevertVB () : HRESULT

SetClass (sProgId : BSTR) : HRESULT

get_nCount (nCount : short*) : HRESULT

Returns the count of properties in the property set.

CopyTo (pDest : IPropertyStorageVB *) : HRESULT

Copies the contents of the source property set to a destination property set.

GetName (nIndex : short, sName : BSTR*) : HRESULT

Returns the name of the specified property.

POLESS.IStorageAmalgamated

Public Operations:

StorageAdd (oStorage : IStorage, bEnd : VARIANT_BOOL = TRUE) :

Adds a new storage to the collection of storages.

ClearStorage () :

Clears all the storage objects from the collection.

POLESS.IStorageVB

Public Operations:

Clear () :

Clears the storage of all elements: sub-storages and streams.

CommitVB () :

Ensures that any changes made to a storage object open in transacted mode are reflected in the parent storage. For nonroot storage objects in direct mode, this method has no effect. For a root storage, it reflects the changes in the actual device, for example, a file on disk. For a root storage object opened in direct mode, always call the Commit method prior to Releasing the object. Commit flushes all memory buffers to the disk for a root storage in direct mode and will return an error code upon failure. Although Releasing the object also flushes memory buffers to disk, it has no capacity to return any error codes upon failure. Therefore, calling Releasing without first calling Commit causes indeterminate results.

RevertVB () :

Discards all changes that have been made to the storage object since the last commit operation.

sElementName (vElement : Variant) : String

Returns the name of the element.

bStorage (vElement : Variant) : Boolean

Returns TRUE if the element is a sub-storage

oElement (vElement : Variant) : Object

Returns either POLESS.iStreamVB or POLESS.iStorageVB for the selected element.

CreateStream (sName : String, eAccess : eACCESS_MODE, bCompression : Boolean) : POLESS.iStreamVB

Creates and opens a stream object with the specified name contained in this storage object. All elements within a storage object — both streams and other storage objects — are kept in the same name space.

Nothing is return if the stream cannot be created.

OpenStream (sName : String, eAccess : eACCESS_MODE) : POLESS.iStreamVB

Opens an existing stream object within this storage object in the specified access mode.

If the stream name is not found, we will look for a stream name prefixed with "\03". This is a linked stream. The contents of this stream is the file and storage where to find this stream.

Nothing is return if the stream cannot be opened.

CreateStorage (sName : String, eAccess : eACCESS_MODE) : POLESS.iStorageVB

Creates and opens a new storage object nested within this storage object.

Nothing is return if the storage cannot be created.

OpenStorage (sName : String, eAccess : eACCESS_MODE) : POLESS.iStorageVB

Opens an existing storage object with the specified name in the specified access mode.

Nothing is return if the storage cannot be opened.

get_sName (sName : BSTR*) : HRESULT

Returns the name of the storage.

get_oStorage (ppDisp : IDispatch) : HRESULT**

Returns the IDispatch interface.

get_nCount (pnCount : short*) : HRESULT

Returns the count of elements in the storage.

GetCompression () : Boolean

Determine if streams may be compressed in the file. If enabled streams may optionally be compressed when created.

GetEncryption () : Boolean

Determine if encryption is enabled for the file. If enabled all streams will be encrypted.

GetCRC () : Boolean

Indicates whether a CRC check is to be performed on the file.

CreateStreamLinked (sName : BSTR, sLocation : BSTR, sFile : BSTR, eAccess : eACCESS_MODE, ppStreamVB : IStreamVB) : HRESULT**

CreatePropertyStg (sName : BSTR, grfFlags : DWORD, bCompress : VARIANT_BOOL, ppPropStg : IPropertyStorageVB) : HRESULT

Create a property storage.

OpenPropertyStg (sName : BSTR, grfFlags : DWORD, dwReserved : DWORD, ppPropStg : IPropertyStorageVB) : HRESULT

SetClass (sProgId : BSTR) : HRESULT

RegisterAlias (sName : BSTR) : HRESULT

Destroy (sName : BSTR) : HRESULT

Destroys the specified element.

POLESS.IStreamVB

Public Operations:

ReadVB (bytData : byte(), nBytes : Integer) : HRESULT

Read data from the stream.

WriteVB (bytData : byte()) : HRESULT

Write data to the stream. The entire byte array is written.

Clear () : HRESULT

Clears the stream of all data.

Reset () : HRESULT

Reset the position in the stream to the begining.

get_sName (sName : BSTR*) : HRESULT

Returns the name of the stream.

get_oStream (ppDisp : IDispatch) : HRESULT**

Returns the IDispatch interface.

CopyTo (pDest : IStreamVB*) : HRESULT

Copies the contents of a source stream to a destination stream.

POLESS.IcStorage

Derived from IStorage

Public Operations:

CreatePropertyStg (sName : BSTR, grfFlags : DWORD, bCompress : Boolean, ppPropStg : IPropertyStorage) : HRESULT**

Creates and opens a property set in a stream object.

OpenPropertyStg (sName : BSTR, grfFlags : DWORD, dwReserved : DWORD, ppPropStg : IPropertyStorage) : HRESULT**

Opens an existing property set in a specified stream object.

CreateStreamLinked (sName : BSTR, sLocation : BSTR, sFile : BSTR, eAccess : eAccess_MODE, ppStream : IStream) : HRESULT**

RegisterAlias (sName : BSTR) : HRESULT

POLESS.IcStorageRoot

Derived from POLESS.IcStorage

Public Operations:

get_Compression (pbCompress : VARIANT_BOOL*) : HRESULT

Determine if streams may be compressed in the file. If enabled streams may optionally be compressed when created.

get_Encryption (pbEncrypt : VARIANT_BOOL*) : HRESULT

Determine if encryption is enabled for the file. If enabled all streams will be encrypted.

get_CRC () : HRESULT

Indicates whether a CRC check is to be performed on the file.

GetObjectFromPath (sItemPath : BSTR, eAccessMode : eACCESS_MODE, refiid : REFIID, ppUnk : IUnknown) : HRESULT**

POLESS.cCrypto

The class control the configuration of the encyrption/decryption of the structured storage.

Public Attributes:

sProviderName : String = MS_DEF_PROV

The name of the Cyrto provider.

eProviderType : ePROVIDER_TYPE = PROV_RSA_FULL

The type of crypto provider.

Cryptographic Provider Types
The field of cryptography is large and growing. There are many different standard data formats and protocols. These are generally organized into groups or families, each of which has its own set of data formats and way of doing things. Even if two families use the same algorithm (for example, the RC2 block cipher), they will often use different padding schemes, different key lengths, and different default modes. Microsoft® CryptoAPI is designed so that a CSP provider type represents a particular family.

ePROV_RSA_FULL
ePROV_RSA_SIG
ePROV_RSA_SCHANNEL
ePROV_DSS
ePROV_DSS_DH
ePROV_DH_SCHANNEL
ePROV_FORTEZZA
ePROV_MS_EXCHANGE
ePROV_SSL

sContainerName : String

Key name. No default, must be provided by used.

sPassword : String

Optional password on the public/private key pair. Only for entery by a human. Can be used for Review disks and their resource files.

POLESS.cFileRoot

The root POLESS class. Must be instanced to perform any POLESS functions. Handles creating and opening POLESS files.

POLESS.cPropertyStorage

POLESS.cPropertyStorageAmalgamated

POLESS.cStorage

POLESS implementation of iStorage. It handles anything POLESS specific and then deligates work to OLE2 compound document Storage class.

POLESS.cStorageAmalgamated

POLESS implementation of iStorage. The class hold references to an order collection of iStorage objects.

When a stream is opened it search the collection of storage objects in order to find the first storage object that has the requested stream. It returns this stream.

It handles compund stream resolution and deligates all other work to POLESS.cStorage. This storage is read-only. It will not allow stream or storages to be created. It primaryly for reading the exam resource file.

Note: This has nothing to do with compund documents.

POLESS.cStorageRoot

POLESS implementation of iStorage and iStorageRoot. It handles anything root POLESS specific and then deligates work to the standard POLESS Storage class.

POLESS.cStream

POLESS implementation of iStream. It handles anything POLESS specific and then deligates work to OLE2 compound document stream class. The specific work includes compression/decompression and encryption/decryption of the stream.

POLESS.iStorageVB

A VB friendly storage interface

POLESS.iStreamVB

A VB friendly stream interface

TOTALS:

1 Logical Packages

26 Classes

LOGICAL PACKAGE STRUCTURE

Logical View
    OLE2SS

What is claimed:

1. A system for computer-based testing comprising:

(a) delivery storage means for storing at least one computer-based test and delivering over an electronic data network said computer-based test to at least one test candidate;

(b) test driver means for controlling delivery of said computer-based test from said delivery storage means; and

(c) cache storage means for caching said computer-based test in response to monitoring of at least one testing environment variable.

2. The computer-based testing system of claim 1, wherein said delivery storage means comprises a plurality of test distribution servers.

3. The computer-based testing system of claim 2, wherein said plurality of test distribution servers comprises:

(a) program deployment means for storing said test driver means for delivery to said test candidate;

(b) data object deployment means for storing cacheable data objects of said computer-based test for delivery to said test candidate; and

(c) application object deployment means for storing cacheable application objects of said computer-based test for delivery to said test candidate.

4. The computer-based testing system of claim 3, wherein said cacheable data objects comprise text, multimedia and template objects.

5. The computer-based testing system of claim 3, wherein said cacheable application objects comprise plugin program objects.

6. The computer-based testing system of claim 1, wherein said cache storage means comprises first and second storage means.

7. The computer-based testing system of claim 6, wherein said first storage means comprises magnetic storage medium.

8. The computer-based testing system of claim 6, wherein said second storage means comprises random access memory (RAM).

9. The computer-based testing system of claim 1, wherein said test driver means comprises:

(a) a session management component for administering said computer-based test to said test candidate;

(b) a proctor authentication component for identifying at least one proctor of said computer-based test;

(c) a scheduling component for verifying test candidate registration for said computer-based test;

(d) a test driver program for controlling delivery of said computer-based test;

(e) a first cache controller for controlling storing of cacheable data objects in encrypted RAM;

(f) a second cache controller for controlling storing of cacheable application objects in RAM; and

(g) a browser presentation component for serving said computer-based test to said test candidate.

10. The computer-based testing system of claim 9, wherein said test driver program comprises:

(a) a first request interface for requesting cacheable data objects;

(b) a second request interface for requesting cacheable application objects;

(c) a request processor for processing requests of said test driver program to retrieve cacheable data objects and cacheable application objects;

(d) a decryption module for decrypting cacheable data objects and cacheable application objects;

(e) a decompression module for decompressing cacheable data objects and cacheable application objects;

(f) a first request module for retrieving cacheable data objects;

(g) a second request module for retrieving cacheable application objects;

(h) a first cache for storing retrieved cacheable data objects;

(i) a second cache for storing retrieved cacheable application objects;

(j) a stimuli processor for adapting delivery of said computer-based test;

(k) a cache controller for controlling source of retrieving cacheable data objects and cacheable application objects and for controlling volume of cacheable data objects and cacheable application objects stored in cache storage means;

(l) a network interface for communicating with a plurality of test distribution servers; and

(m) an authentication component for authenticating cacheable data objects and cacheable application objects.

11. The computer-based testing system of claim 1, wherein said test driver means comprises at least one monitoring means for monitoring a testing environment variable.

12. The computer-based testing system of claim 11, wherein said monitoring means is a test candidate progress monitor for measuring progress of said test candidate during computer-based testing.

13. The computer-based testing system of claim 11, wherein said monitoring means is a test candidate performance monitor for measuring competency of said test candidate during computer-based testing.

14. The computer-based testing system of claim 11, wherein said monitoring means is a network bandwidth monitor

for determining a speed of a network connection through which said computer-based test is being delivered.

15. The computer-based testing system of claim 11, wherein said monitoring means is a network state monitor for determining a state of a network connection through which said computer-based test is being delivered.

16. The computer-based testing system of claim 11, wherein said monitoring means is a server state monitor for determining a state of a server from which said computer-based test is being delivered.

17. The computer-based testing system of claim 1, wherein said electronic data network is the Internet.

18. A method for computer-based testing comprising the steps of:

(a) storing at least one computer-based test and delivering over an electronic data network said computer-based test to at least one test candidate by delivery storage means;

(b) controlling delivery of said computer-based test from said delivery storage means by test driver means; and

(c) caching and storing said computer-based test in cache storage means in response to monitoring of at least one testing environment variable.

19. The method for computer-based testing of claim 18, the method comprising the step of storing on a plurality of test distribution servers said computer-based test for delivery to a plurality of test candidates.

20. The method for computer-based testing of claim 19, the method comprising the step of storing cacheable data objects comprising text, multimedia and template objects.

21. The method for computer-based testing of claim 19, the method comprising the step of storing cacheable application objects comprising plugin program objects.

22. The method for computer-based testing of claim 19, the method comprising the step of storing said test driver means in first storage means comprising magnetic storage medium.

23. The method for computer-based testing of claim 18, the method comprising the step of storing said cacheable

data objects in second storage means comprising random access memory (RAM).

24. The method for computer-based testing of claim 18, the method comprising the step of storing said cacheable application objects in second storage means comprising random access memory (RAM).

25. The method for computer-based testing system of claim 18, the method comprising the steps of:

(a) administering said computer-based test to said test candidate using a session management component;

(b) identifying at least one proctor of said computer-based test using a proctor authentication component;

(c) verifying test candidate registration for said computer-based test using a scheduling component;

(d) controlling delivery of said computer-based test using a test driver program;

(e) controlling storing of cacheable data objects in encrypted RAM using a first cache controller;

(f) controlling storing of cacheable application objects in RAM using a second cache controller; and

(g) serving said computer-based test to said test candidate using a browser presentation component.

26. The method of computer-based testing of claim 25, the method comprising the steps of:

(a) requesting cacheable data objects using a first request interface;

(b) requesting cacheable application objects a second request interface;

(c) processing requests of said test driver program to retrieve cacheable data objects and cacheable application objects using a request processor;

(d) decrypting cacheable data objects and cacheable application objects using a decryption module;

(e) decompressing cacheable data objects and cacheable application objects using a decompression module;

(f) retrieving cacheable data objects using a first request module;

(g) retrieving cacheable application objects using a second request module;

(h) storing retrieved cacheable data objects using a first cache;

(i) storing retrieved cacheable application objects using a second cache;

(j) adapting delivery of said computer-based test using a stimuli processor;

(k) controlling source for retrieval of cacheable data objects and cacheable application objects and for controlling volume of cacheable data objects and cacheable application objects stored in cache storage means using a cache controller;

(l) communicating with a plurality of test distribution servers using a network interface; and

(m) authenticating cacheable data objects and cacheable application objects using an authentication component.

27. The method of computer-based testing of claim 18, the method comprising the step of monitoring at least one test environment variable.

28. The method of computer-based testing of claim 27, the method comprising the step of monitoring test candidate progress for measuring progress of said test candidate during computer-based testing.

29. The method of computer-based testing of claim 27, the method comprising the step of monitoring test candidate performance for measuring competency of said test candidate during computer-based testing.

30. The method of computer-based testing of claim 27, the method comprising the step of monitoring network bandwidth for determining a speed of a network connection through which said computer-based test is being delivered.

31. The method of computer-based testing of claim 27, the method comprising the step of monitoring network state for determining a state of a network connection through which said computer-based test is being delivered.

32. The method of computer-based testing of claim 27, the method comprising the step of monitoring server state for determining a state of a server from which said computer-based test is being delivered.

33. The method for computer-based testing of claim 18, the method comprising the step of delivering over the Internet said computer-based test.

34. A computer-readable storage medium for storing a computer-based testing method, said computer-based testing method comprising:

(a) storing at least one computer-based test on delivery storage means and delivering over an electronic data network said computer-based test to at least one test candidate by said delivery storage means;

(b) controlling delivery of said computer-based test from said delivery storage means by test driver means; and

(c) caching and storing said computer-based test in cache storage means in response to monitoring of at least one testing environment variable.

35. The storage medium for storing a computer-based testing method of claim 34, the method comprising the step of storing on a plurality of test distribution servers said computer-based test for delivery to a plurality of test candidates.

36. The storage medium for storing a computer-based testing method of claim 35, the method comprising the step of storing cacheable data objects comprising text, multimedia and template objects.

37. The storage medium for storing a computer-based testing method of claim 35, the method comprising the step of storing cacheable application objects comprising plugin program objects.

38. The storage medium for storing a computer-based testing method of claim 35, the method comprising the step of storing said test driver means in first storage means comprising magnetic storage medium.

39. The storage medium for storing a computer-based testing method of claim 35, the method comprising the step of storing said cacheable data objects in second storage means comprising random access memory (RAM).

40. The storage medium for storing a computer-based testing method of claim 35, the method comprising the step of storing said cacheable application objects in second storage means comprising random access memory (RAM).

41. The storage medium for storing a computer-based testing method of claim 34, the method comprising the steps of:

(a) administering said computer-based test to said test candidate using a session management component;

(b) identifying at least one proctor of said computer-based test using a proctor authentication component;

(c) verifying test candidate registration for said computer-based test using a scheduling component;

(d) controlling delivery of said computer-based test using a test driver program;

(e) controlling storing of cacheable data objects in encrypted RAM using a first cache controller;

(f) controlling storing of cacheable application objects in RAM using a second cache controller; and

(g) serving said computer-based test to said test candidate using a browser presentation component.

42. The storage medium for storing a computer-based testing method of claim 41, the method comprising the steps of:

(a) requesting cacheable data objects using a first request interface;

(b) requesting cacheable application objects a second request interface;

(c) processing requests of said test driver program to retrieve cacheable data objects and cacheable application objects using a request processor;

(d) decrypting cacheable data objects and cacheable application objects using a decryption module;

(e) decompressing cacheable data objects and cacheable application objects using a decompression module;

(f) retrieving cacheable data objects using a first request module;

(g) retrieving cacheable application objects using a second request module;

(h) storing cacheable data objects using a first cache;

(i) storing cacheable application objects using a second cache;

(j) adapting delivery of said computer-based test using a stimuli processor;

(k) controlling source for retrieval of cacheable data objects and cacheable application objects and for controlling volume of cacheable data objects and cacheable application objects stored in cache storage means using a cache controller;

(l) communicating with a plurality of test distribution servers using a network interface; and

(m) authenticating cacheable data objects and cacheable application objects using an authentication component.

43. The storage medium for storing a computer-based testing method of claim 34, the method comprising the step of monitoring at least one test environment variable.

44. The storage medium for storing a computer-based testing method of claim 43, the method comprising the step of monitoring test candidate progress for measuring progress of said test candidate during computer-based testing.

45. The storage medium for storing a computer-based testing method of claim 43, the method comprising the step of monitoring test candidate performance for measuring competency of said test candidate during computer-based testing.

46. The storage medium for storing a computer-based testing method of claim 43, the method comprising the step of monitoring network bandwidth for determining a speed of a network connection through which said computer-based test is being delivered.

47. The storage medium for storing a computer-based testing method of claim 43, the method comprising the step of monitoring network state for determining a state of a

network connection through which said computer-based test is being delivered.

48. The storage medium for storing a computer-based testing method of claim 43, the method comprising the step of monitoring server state for determining a state of a server from which said computer-based test is being delivered.

49. The storage medium for storing a computer-based testing method of claim 43, the method comprising the step of delivering over the Internet said computer-based test.

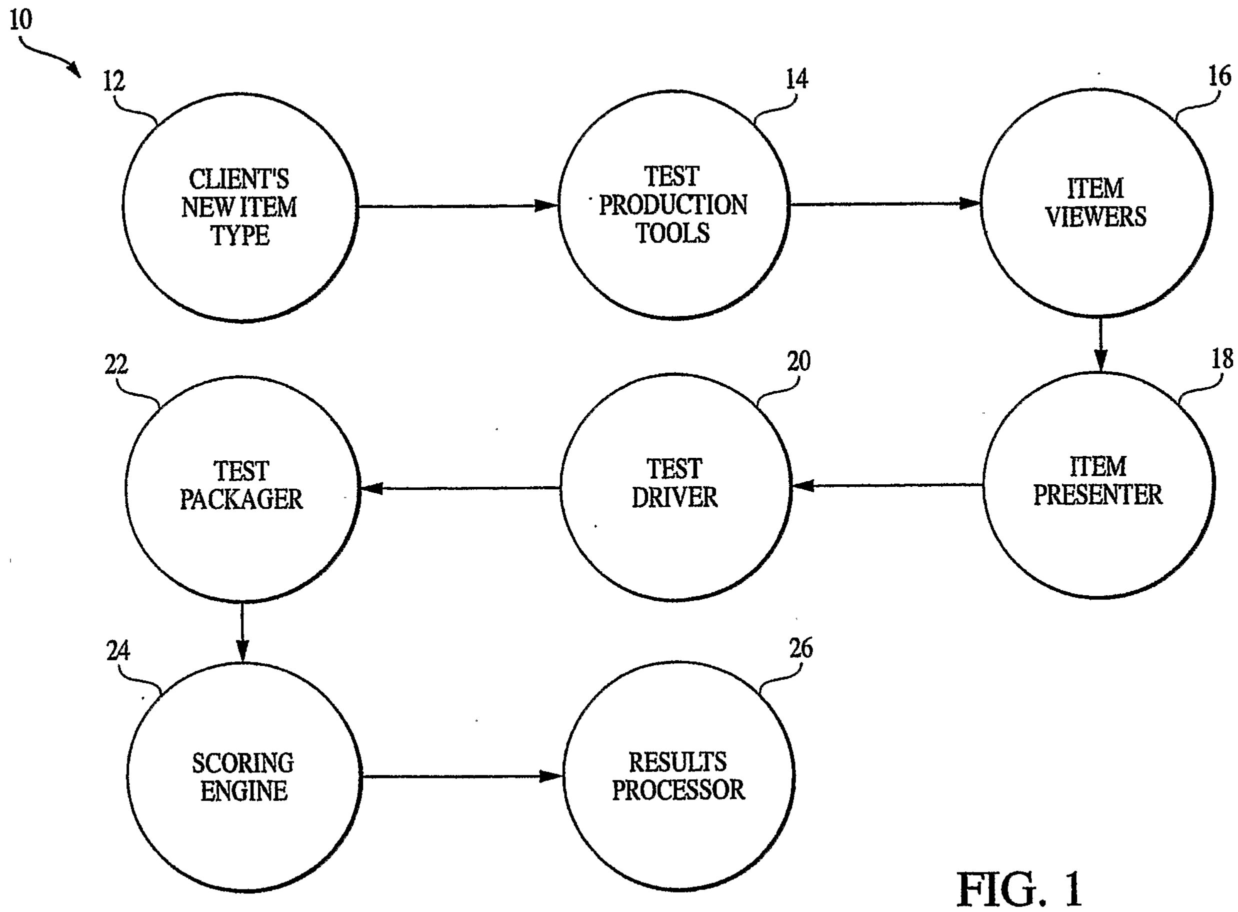


FIG. 1
(PRIOR ART)

FUNCTIONS, AREAS, AND SOFTWARE FOR TEST PREPARATION/ASSEMBLY

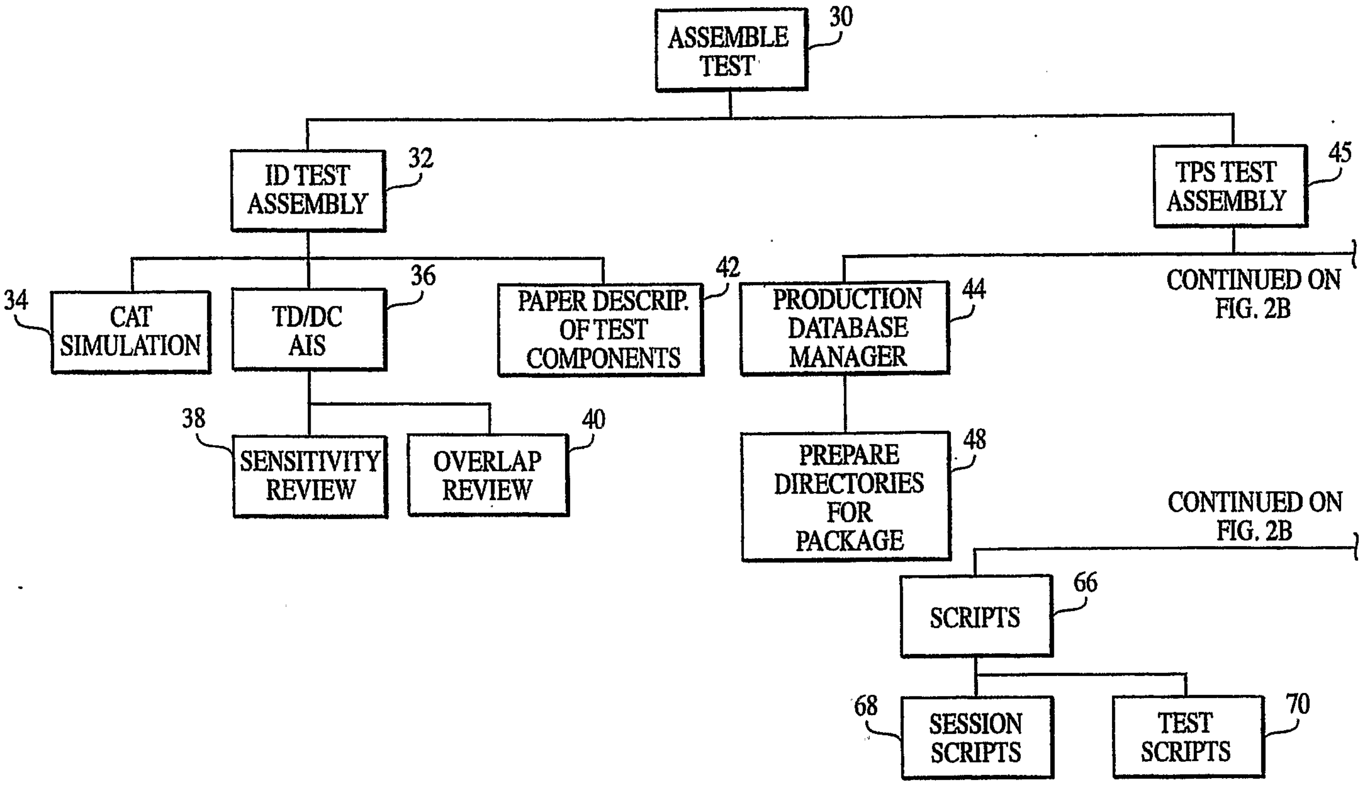


FIG. 2A
(PRIOR ART)

FUNCTIONS, AREAS, AND SOFTWARE FOR TEST PREPARATION/ASSEMBLY
(CONTINUED)

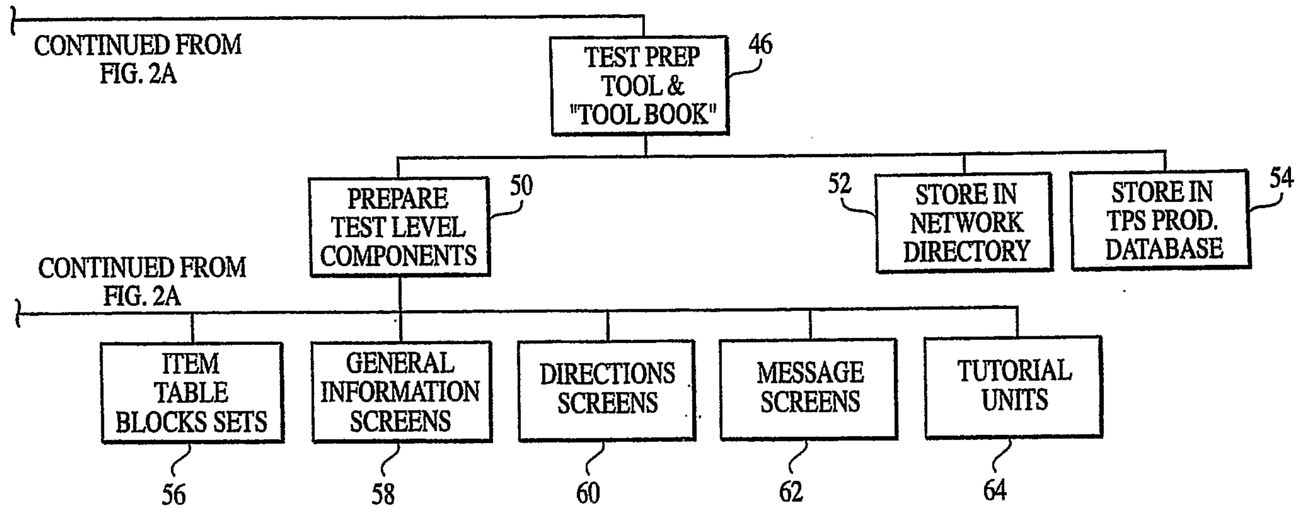

CONTINUED FROM FIG. 2A
TEST PREP TOOL & "TOOL BOOK"
46
PREPARE TEST LEVEL COMPONENTS
50
52
STORE IN NETWORK DIRECTORY
STORE IN TPS PROD. DATABASE
54
CONTINUED FROM FIG. 2A
ITEM TABLE BLOCKS SETS
GENERAL INFORMATION SCREENS
DIRECTIONS SCREENS
MESSAGE SCREENS
TUTORIAL UNITS
56
58
60
62
64

FIG. 2B
(PRIOR ART)

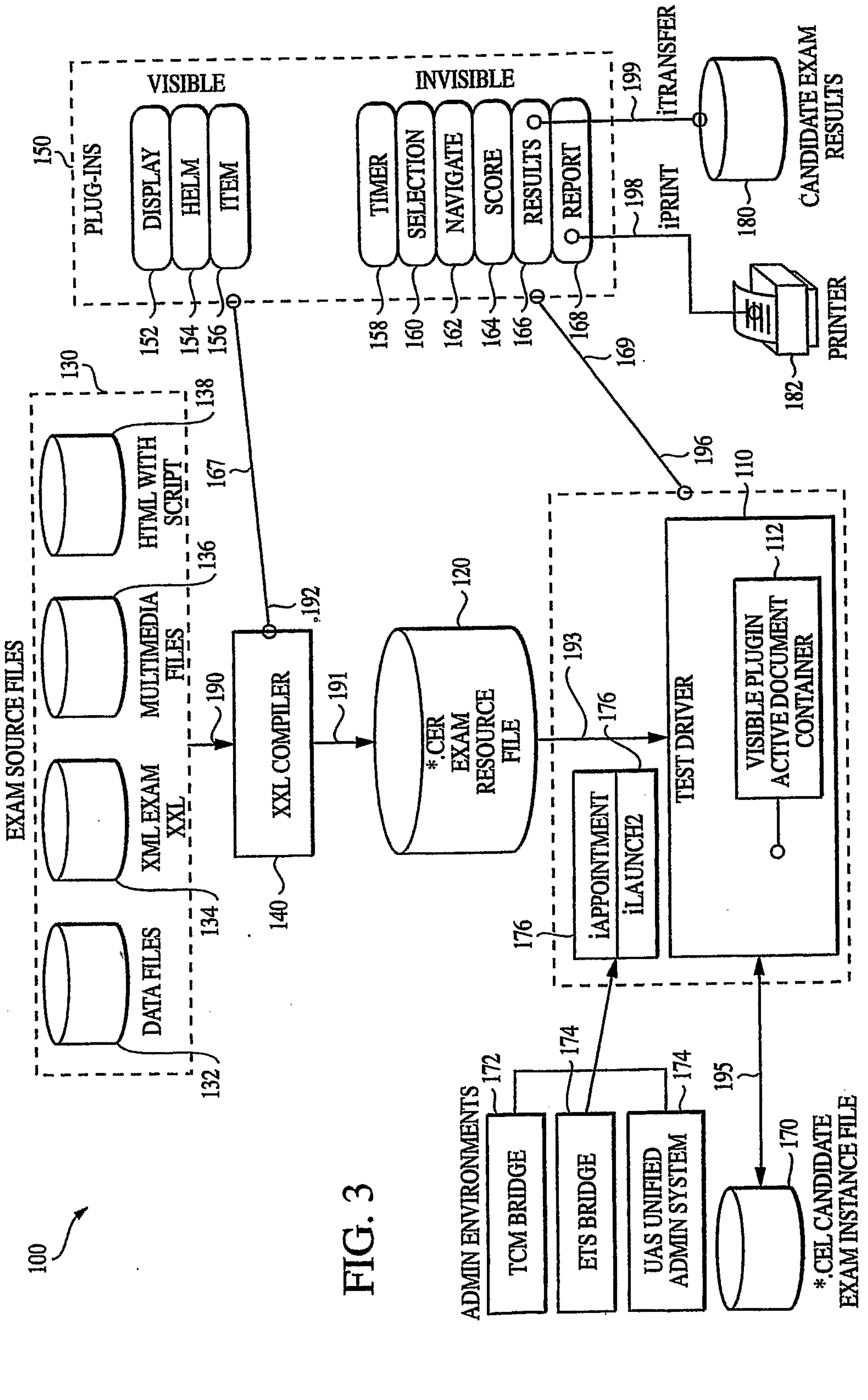

100
EXAM SOURCE FILES
130
DATA FILES
132
XML EXAM XXL
134
MULTIMEDIA FILES
136
HTML WITH SCRIPT
138
190
XXL COMPILER
140
191
192
167
*.CER EXAM RESOURCE FILE
120
193
176
TEST DRIVER
110
VISIBLE PLUGIN ACTIVE DOCUMENT CONTAINER
112
iAPPOINTMENT
iLAUNCH2
176
196
169
150
PLUG-INS
VISIBLE
DISPLAY
HELM
ITEM
152
154
156
INVISIBLE
TIMER
SELECTION
NAVIGATE
SCORE
RESULTS
REPORT
158
160
162
164
166
168
198
iPRINT
199
iTRANSFER
CANDIDATE EXAM RESULTS
180
PRINTER
182
ADMIN ENVIRONMENTS
TCM BRIDGE
172
ETS BRIDGE
174
UAS UNIFIED ADMIN SYSTEM
174
195
*.CEL CANDIDATE EXAM INSTANCE FILE
170

FIG. 3

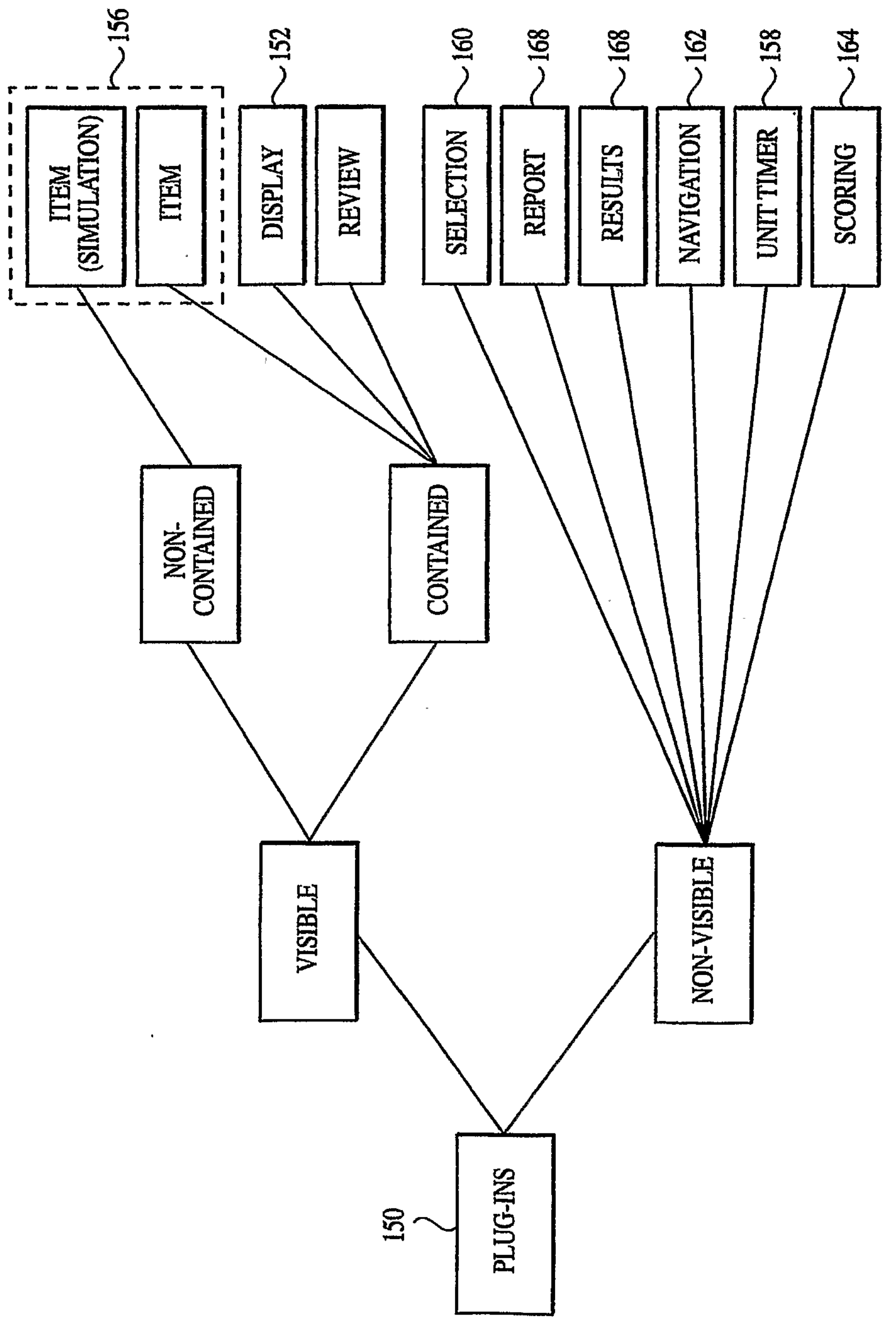

150
PLUG-INS
VISIBLE
NON-VISIBLE
NON-CONTAINED
CONTAINED
156
ITEM (SIMULATION)
ITEM
152
DISPLAY
REVIEW
160
SELECTION
168
REPORT
168
RESULTS
162
NAVIGATION
158
UNIT TIMER
164
SCORING

FIG. 4

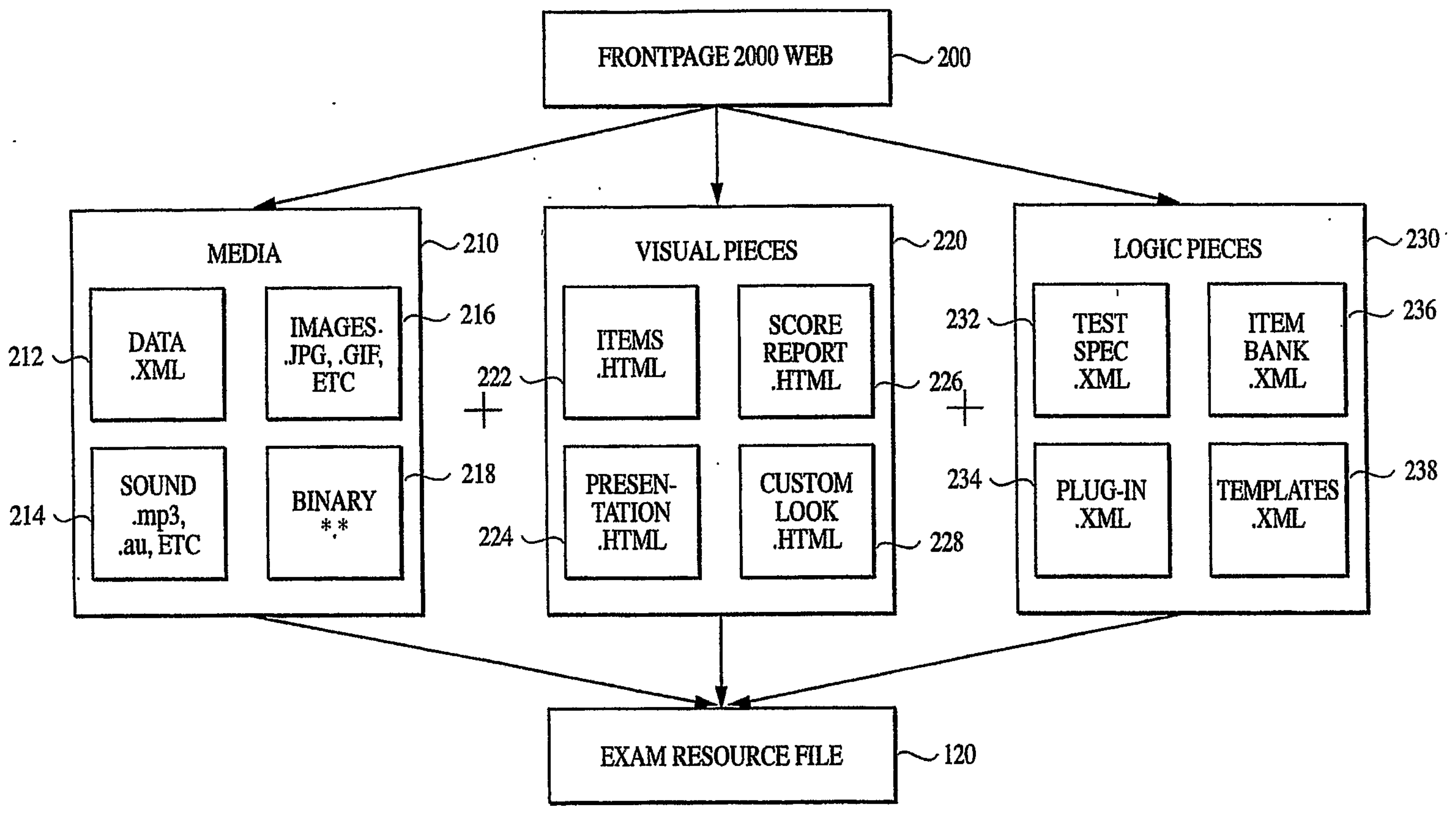
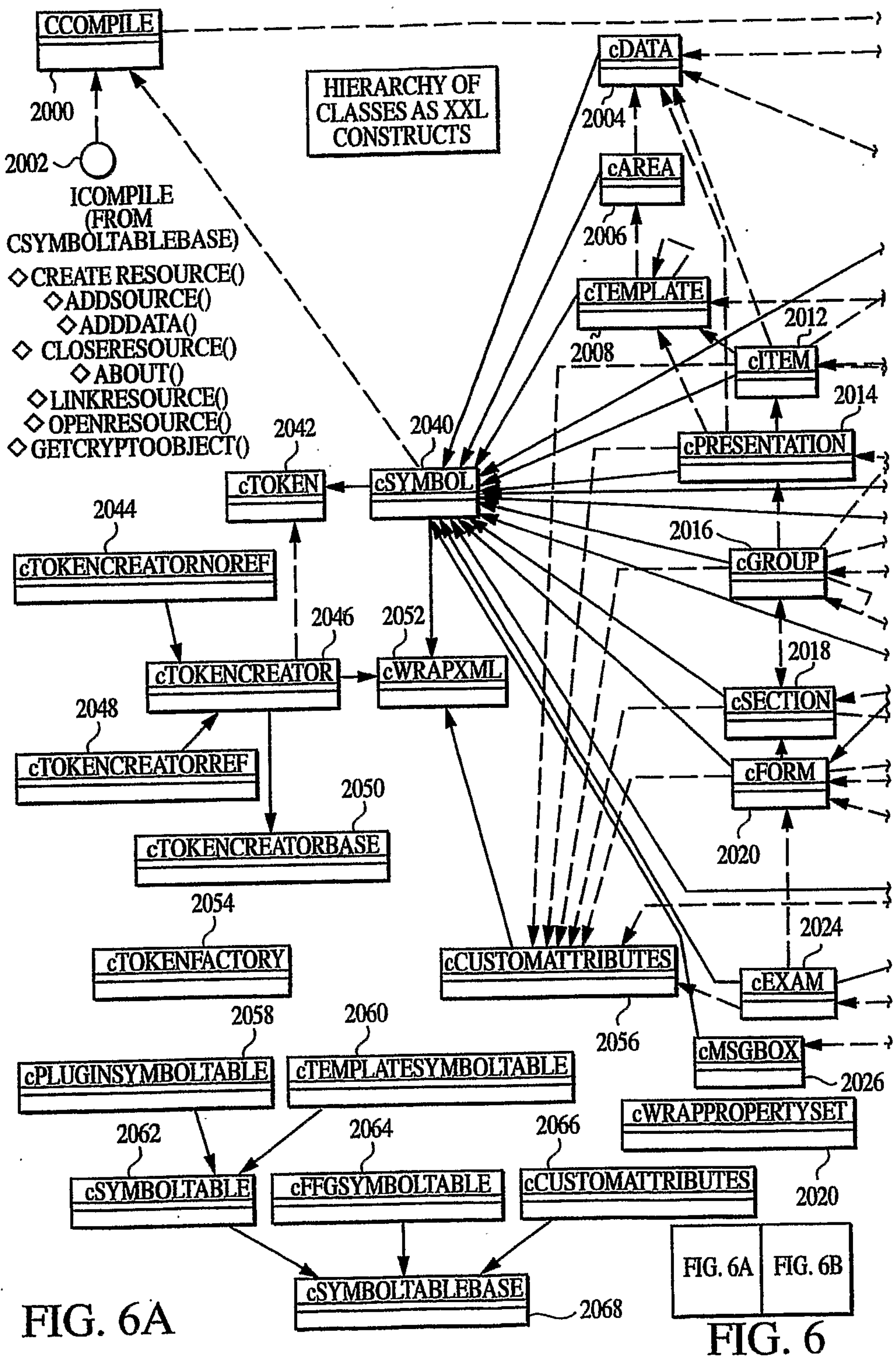

FRONTPAGE 2000 WEB
200
MEDIA
210
DATA .XML
212
IMAGES .JPG, .GIF, ETC
216
SOUND .mp3, .au, ETC
214
BINARY *.*
218
VISUAL PIECES
220
ITEMS .HTML
222
SCORE REPORT .HTML
226
PRESEN-TATION .HTML
224
CUSTOM LOOK .HTML
228
LOGIC PIECES
230
TEST SPEC .XML
232
ITEM BANK .XML
236
PLUG-IN .XML
234
TEMPLATES .XML
238
EXAM RESOURCE FILE
120

FIG. 5



HIERARCHY OF CLASSES AS XXL CONSTRUCTS
CCOMPILE
2000
2002
ICOMPILE (FROM CSYMBOLTABLEBASE)
◇CREATE RESOURCE()
◇ADDSOURCE()
◇ADDDATA()
◇ CLOSERESOURCE()
◇ABOUT()
◇LINKRESOURCE()
◇OPENRESOURCE()
◇GETCRYPTOOBJECT()
cDATA
2004
cAREA
2006
cTEMPLATE
2008
cITEM
2012
cPRESENTATION
2014
cGROUP
2016
cSECTION
2018
cFORM
2020
cEXAM
2024
cMSGBOX
2026
cWRAPPROPERTYSET
2020
cSYMBOL
2040
cTOKEN
2042
cTOKENCREATORNOREF
2044
cTOKENCREATOR
2046
cWRAPXML
2052
cTOKENCREATORREF
2048
cTOKENCREATORBASE
2050
cTOKENFACTORY
2054
cCUSTOMATTRIBUTES
2056
cPLUGINSYMBOLTABLE
2058
cTEMPLATESYMBOLTABLE
2060
cSYMBOLTABLE
2062
cFFGSYMBOLTABLE
2064
cCUSTOMATTRIBUTES
2066
cSYMBOLTABLEBASE
2068
FIG. 6A
FIG. 6B

FIG. 6A

FIG. 6

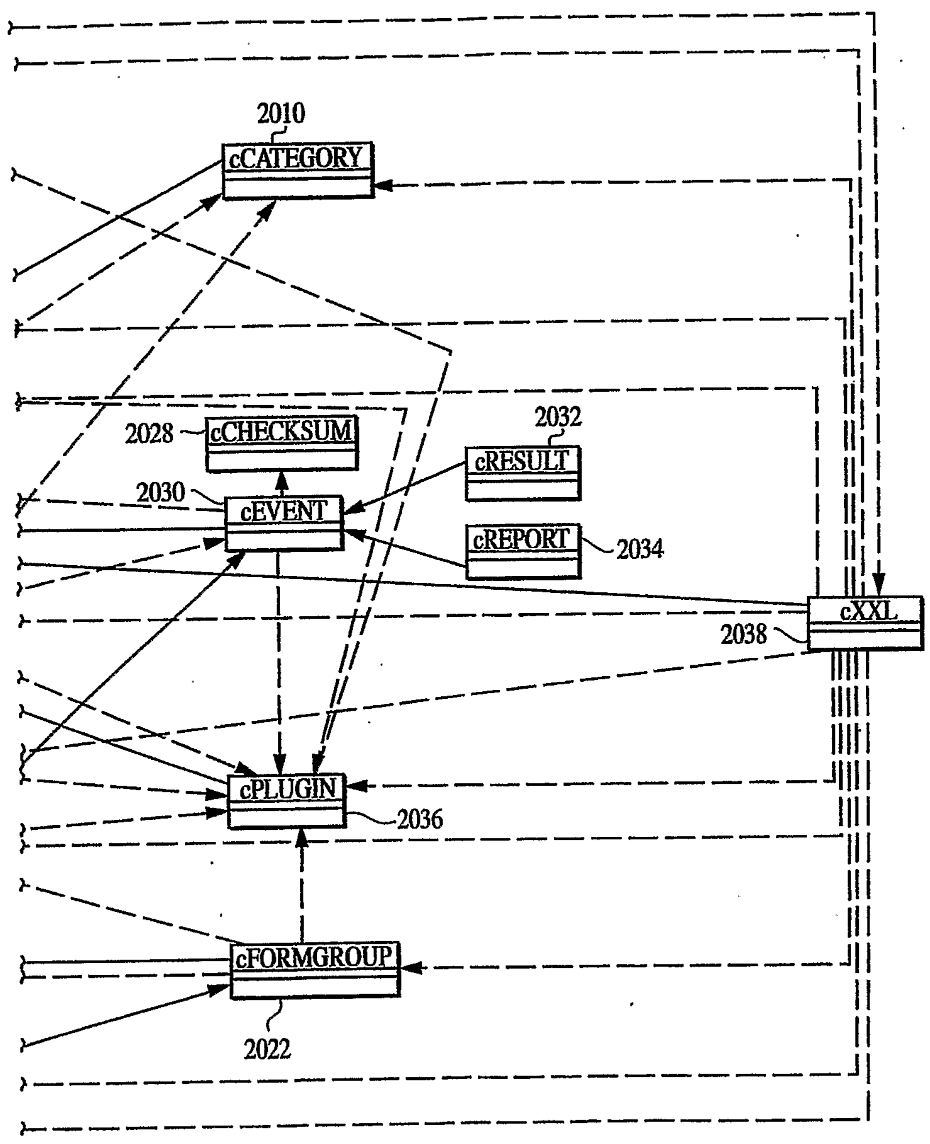

2010
cCATEGORY
2028
cCHECKSUM
2032
cRESULT
2030
cEVENT
cREPORT
2034
cXXL
2038
cPLUGIN
2036
cFORMGROUP
2022

FIG. 6B

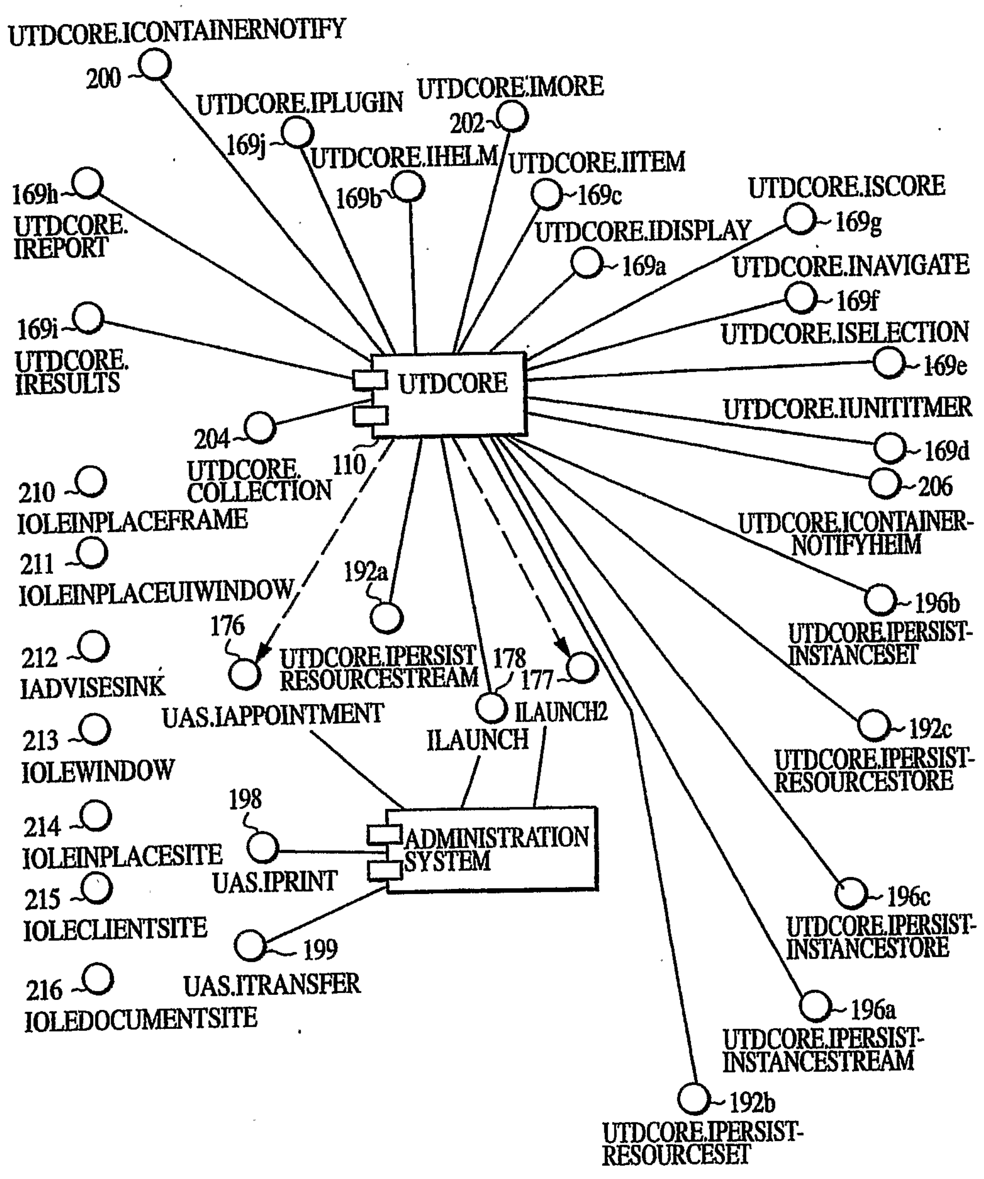

UTDCORE.ICONTAINERNOTIFY
200
UTDCORE.IPLUGIN
169j
UTDCORE.IMORE
202
UTDCORE.IHELM
169b
UTDCORE.IITEM
169c
169h
UTDCORE.IREPORT
UTDCORE.ISCORE
169g
UTDCORE.IDISPLAY
169a
UTDCORE.INAVIGATE
169f
169i
UTDCORE.IRESULTS
UTDCORE.ISELECTION
169e
UTDCORE
204
UTDCORE.COLLECTION
110
UTDCORE.IUNITITMER
169d
206
210
IOLEINPLACEFRAME
211
IOLEINPLACEUIWINDOW
UTDCORE.ICONTAINER-NOTIFYHEIM
192a
196b
176
UTDCORE.IPERSIST-INSTANCESET
212
IADVISESINK
UTDCORE.IPERSIST RESOURCESTREAM
178
177
ILAUNCH2
UAS.IAPPOINTMENT
192c
213
IOLEWINDOW
ILAUNCH
UTDCORE.IPERSIST-RESOURCESTORE
198
214
IOLEINPLACESITE
ADMINISTRATION SYSTEM
UAS.IPRINT
215
IOLECLIENTSITE
196c
UTDCORE.IPERSIST-INSTANCESTORE
199
216
UAS.ITRANSFER
IOLEDOCUMENTSITE
196a
UTDCORE.IPERSIST-INSTANCESTREAM
192b
UTDCORE.IPERSIST-RESOURCESET

FIG. 7

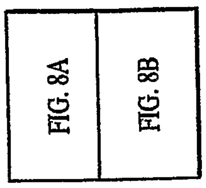


FIG. 8A
FIG. 8B

FIG. 8

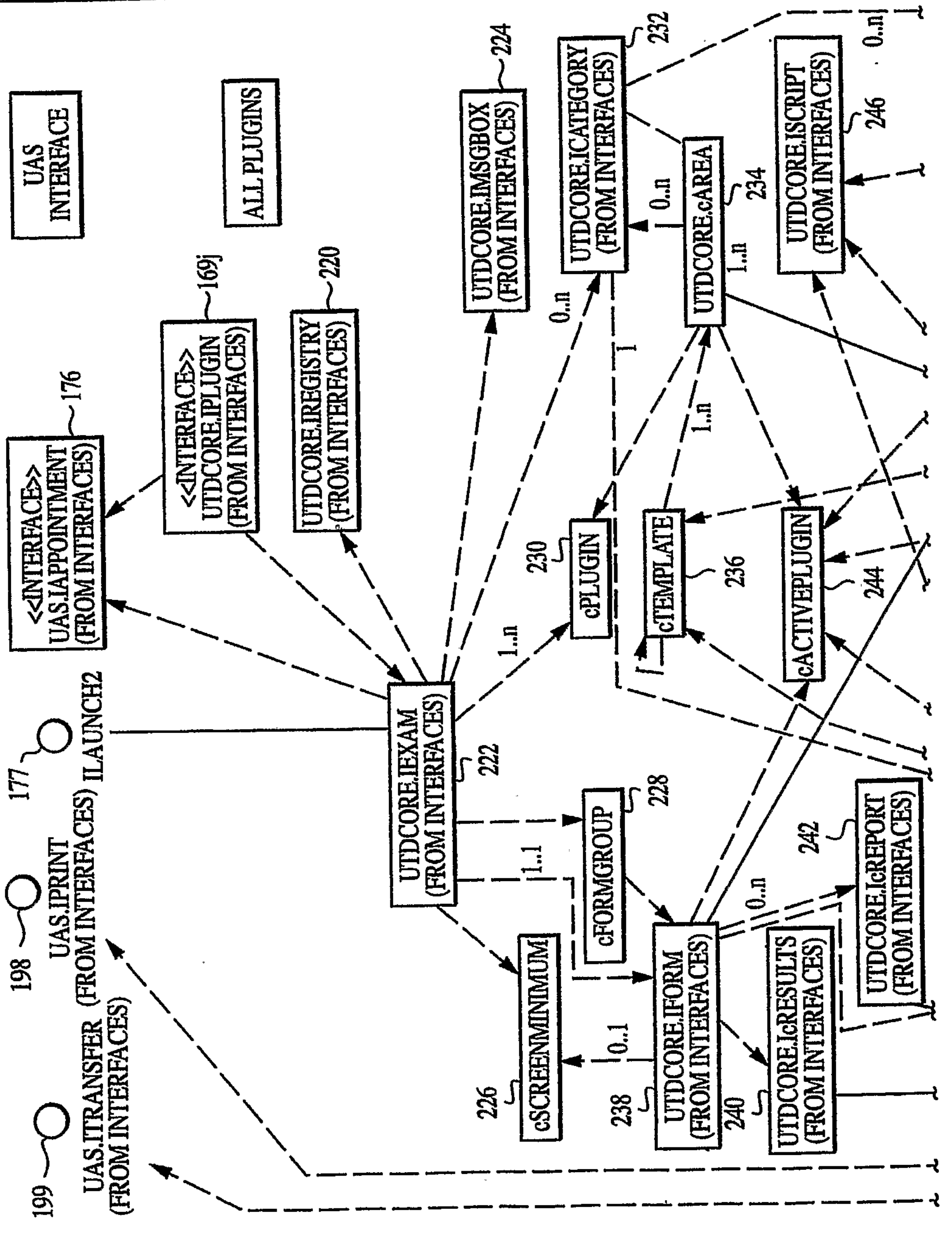
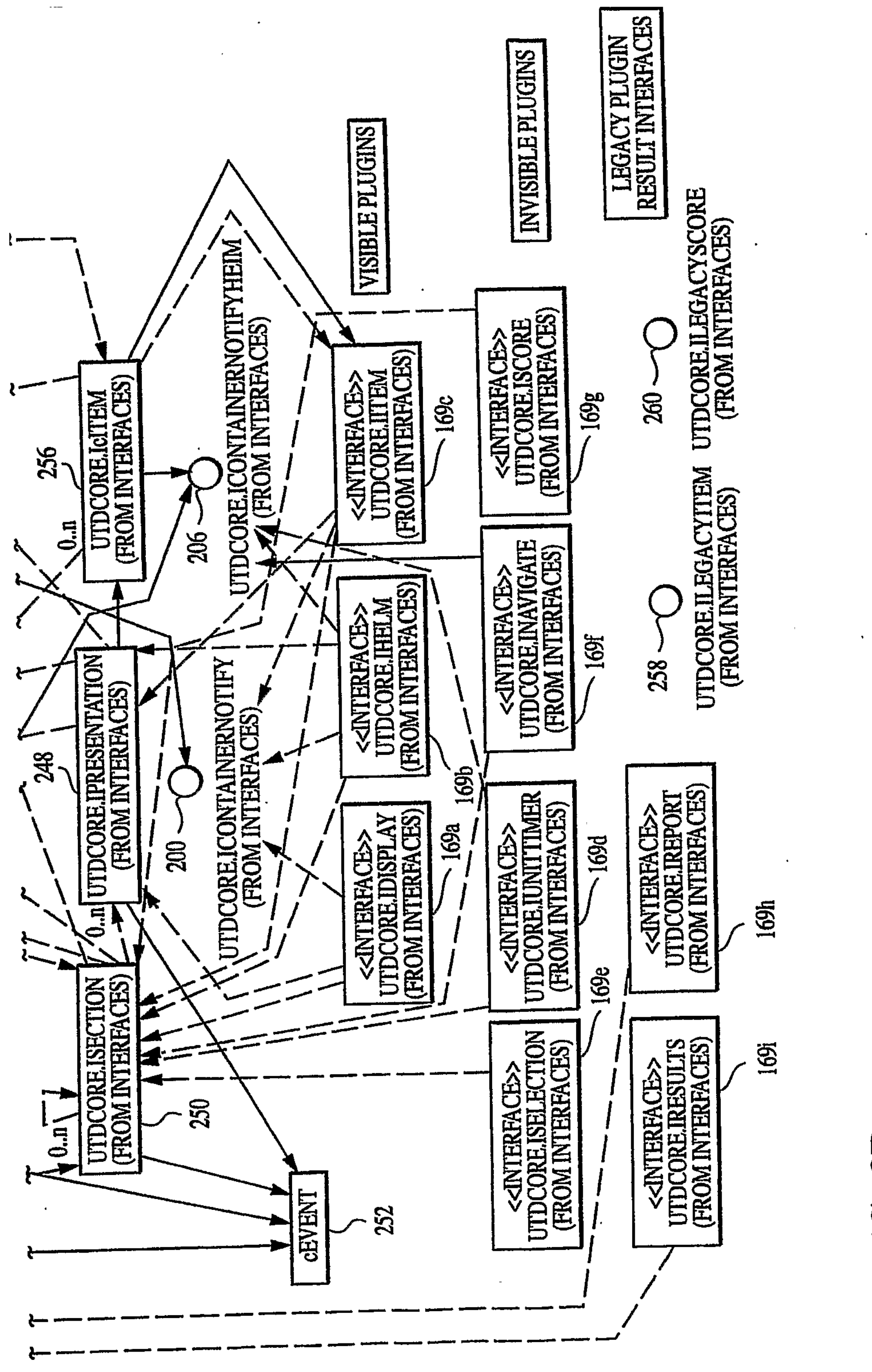

UAS INTERFACE
ALL PLUGINS
<<INTERFACE>> UAS.IAPPOINTMENT (FROM INTERFACES)
176
<<INTERFACE>> UTDCORE.IPLUGIN (FROM INTERFACES)
169j
UTDCORE.IREGISTRY (FROM INTERFACES)
220
UTDCORE.IMSGBOX (FROM INTERFACES)
224
UTDCORE.ICATEGORY (FROM INTERFACES)
232
UTDCORE.cAREA
234
UTDCORE.ISCRIPT (FROM INTERFACES)
246
cPLUGIN
230
cTEMPLATE
236
cACTIVEPLUGIN
244
UTDCORE.IEXAM (FROM INTERFACES)
222
ILAUNCH2
177
UAS.IPRINT (FROM INTERFACES)
198
UAS.ITRANSFER (FROM INTERFACES)
199
cFORMGROUP
228
cSCREENMINIMUM
226
UTDCORE.IFORM (FROM INTERFACES)
238
UTDCORE.IcRESULTS (FROM INTERFACES)
240
UTDCORE.IcREPORT (FROM INTERFACES)
242
0..n
1..n
1
1..1
0..1

FIG. 8A



VISIBLE PLUGINS
INVISIBLE PLUGINS
LEGACY PLUGIN RESULT INTERFACES
UTDCORE.IcITEM (FROM INTERFACES)
256
0..n
206
UTDCORE.ICONTAINERNOTIFYHEIM (FROM INTERFACES)
<<INTERFACE>> UTDCORE.IITEM (FROM INTERFACES)
169c
<<INTERFACE>> UTDCORE.ISCORE (FROM INTERFACES)
169g
260
UTDCORE.ILEGACYSCORE (FROM INTERFACES)
UTDCORE.IPRESENTATION (FROM INTERFACES)
248
200
UTDCORE.ICONTAINERNOTIFY (FROM INTERFACES)
<<INTERFACE>> UTDCORE.IHELM (FROM INTERFACES)
169b
<<INTERFACE>> UTDCORE.INAVIGATE (FROM INTERFACES)
169f
258
UTDCORE.ILEGACYITEM (FROM INTERFACES)
<<INTERFACE>> UTDCORE.IDISPLAY (FROM INTERFACES)
169a
<<INTERFACE>> UTDCORE.IUNITTIMER (FROM INTERFACES)
169d
<<INTERFACE>> UTDCORE.IREPORT (FROM INTERFACES)
169h
0..n
UTDCORE.ISECTION (FROM INTERFACES)
250
0..n
<<INTERFACE>> UTDCORE.ISELECTION (FROM INTERFACES)
169e
<<INTERFACE>> UTDCORE.IRESULTS (FROM INTERFACES)
169i
cEVENT
252

FIG. 8B

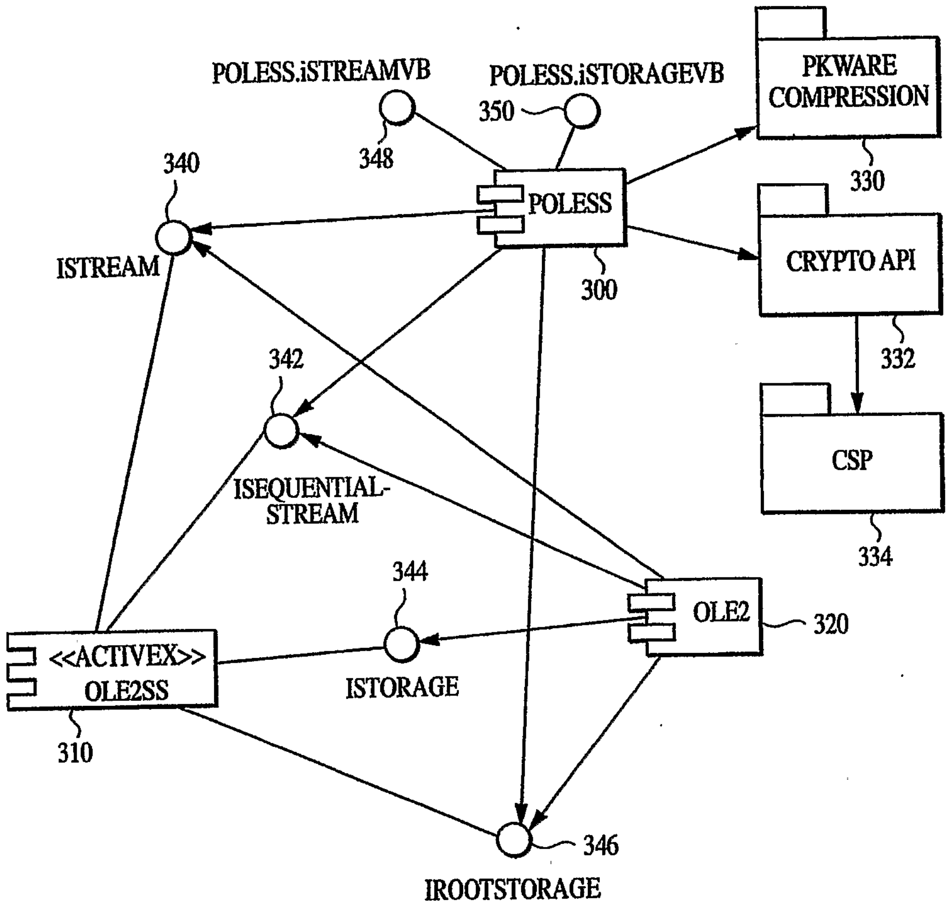
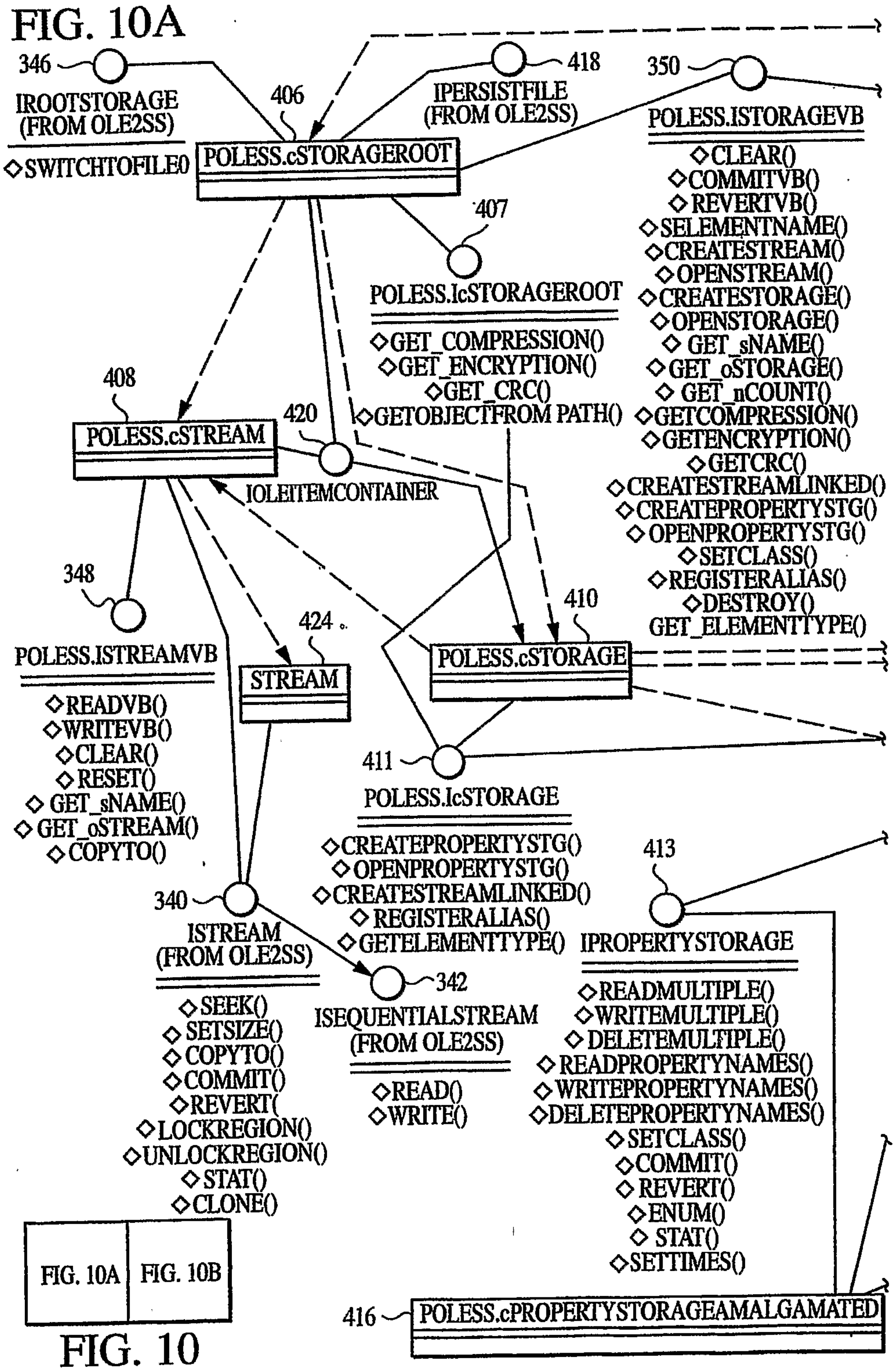

POLESS.iSTREAMVB
348
POLESS.iSTORAGEVB
350
PKWARE COMPRESSION
330
340
ISTREAM
POLESS
300
CRYPTO API
332
342
ISEQUENTIAL-STREAM
CSP
334
344
ISTORAGE
OLE2
320
<<ACTIVEX>> OLE2SS
310
346
IROOTSTORAGE

FIG. 9

FIG. 10A



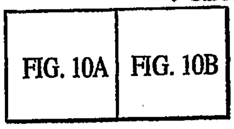


FIG. 10

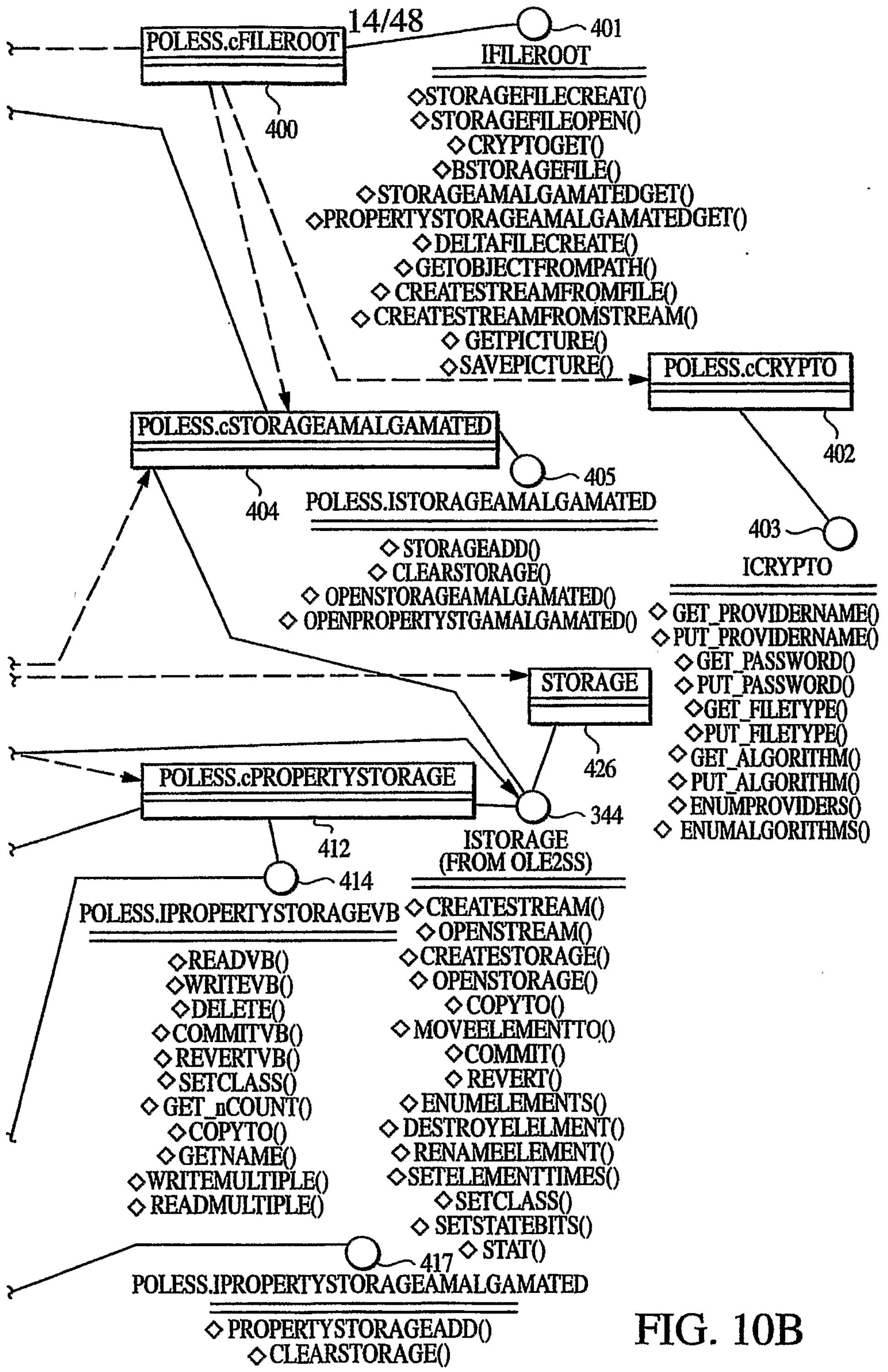

14/48
POLESS.cFILEROOT
400
401
IFILEROOT
◇STORAGEFILECREAT()
◇STORAGEFILEOPEN()
◇CRYPTOGET()
◇BSTORAGEFILE()
◇STORAGEAMALGAMATEDGET()
◇PROPERTYSTORAGEAMALGAMATEDGET()
◇DELTAFILECREATE()
◇GETOBJECTFROMPATH()
◇CREATESTREAMFROMFILE()
◇CREATESTREAMFROMSTREAM()
◇GETPICTURE()
◇SAVEPICTURE()
POLESS.cCRYPTO
402
403
ICRYPTO
◇GET_PROVIDERNAME()
◇PUT_PROVIDERNAME()
◇GET_PASSWORD()
◇PUT_PASSWORD()
◇GET_FILETYPE()
◇PUT_FILETYPE()
◇GET_ALGORITHM()
◇PUT_ALGORITHM()
◇ENUMPROVIDERS()
◇ENUMALGORITHMS()
POLESS.cSTORAGEAMALGAMATED
404
405
POLESS.ISTORAGEAMALGAMATED
◇STORAGEADD()
◇CLEARSTORAGE()
◇OPENSTORAGEAMALGAMATED()
◇OPENPROPERTYSTGAMALGAMATED()
STORAGE
426
POLESS.cPROPERTYSTORAGE
412
344
ISTORAGE
(FROM OLE2SS)
◇CREATESTREAM()
◇OPENSTREAM()
◇CREATESTORAGE()
◇OPENSTORAGE()
◇COPYTO()
◇MOVEELEMENTTO()
◇COMMIT()
◇REVERT()
◇ENUMELEMENTS()
◇DESTROYELELMENT()
◇RENAMEELEMENT()
◇SETELEMENTTIMES()
◇SETCLASS()
◇SETSTATEBITS()
◇STAT()
414
POLESS.IPROPERTYSTORAGEVB
◇READVB()
◇WRITEVB()
◇DELETE()
◇COMMITVB()
◇REVERTVB()
◇SETCLASS()
◇GET_nCOUNT()
◇COPYTO()
◇GETNAME()
◇WRITEMULTIPLE()
◇READMULTIPLE()
417
POLESS.IPROPERTYSTORAGEAMALGAMATED
◇PROPERTYSTORAGEADD()
◇CLEARSTORAGE()

FIG. 10B

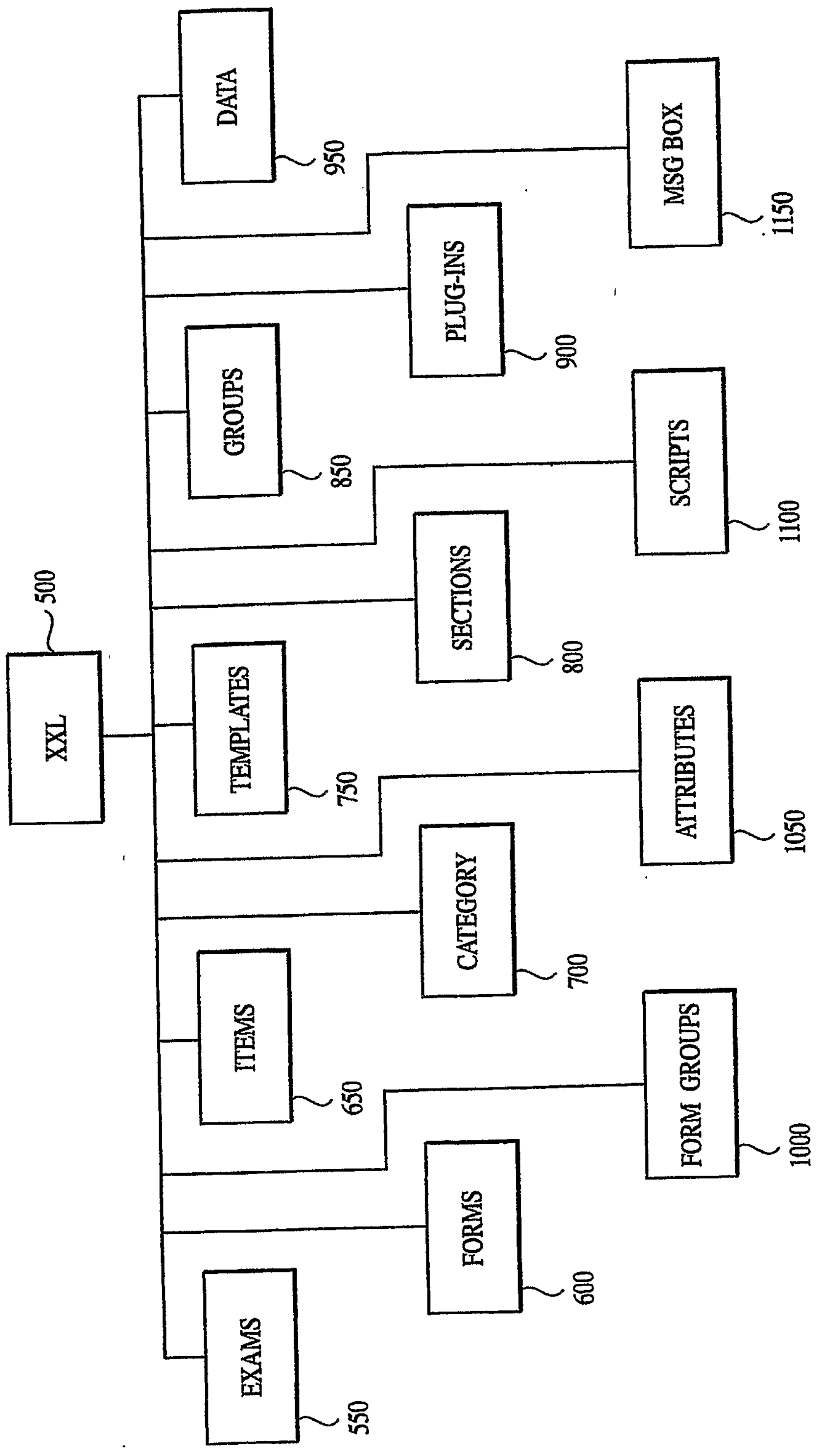

XXL
500
EXAMS
550
FORMS
600
ITEMS
650
CATEGORY
700
TEMPLATES
750
SECTIONS
800
GROUPS
850
PLUG-INS
900
DATA
950
FORM GROUPS
1000
ATTRIBUTES
1050
SCRIPTS
1100
MSG BOX
1150

FIG. 11

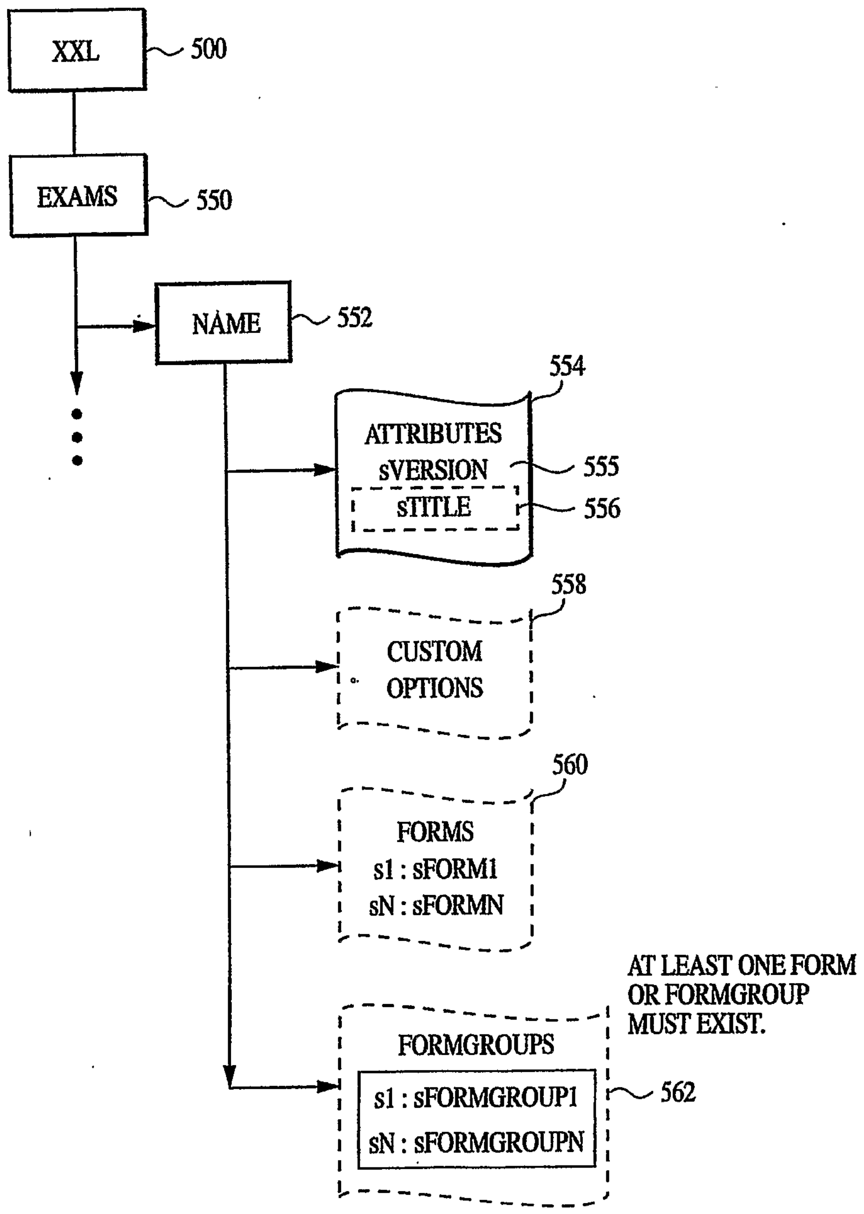

XXL
500
EXAMS
550
NAME
552
554
ATTRIBUTES
sVERSION
555
sTITLE
556
558
CUSTOM OPTIONS
560
FORMS
s1 : sFORM1
sN : sFORMN
AT LEAST ONE FORM OR FORMGROUP MUST EXIST.
FORMGROUPS
s1 : sFORMGROUP1
sN : sFORMGROUPN
562

FIG. 12

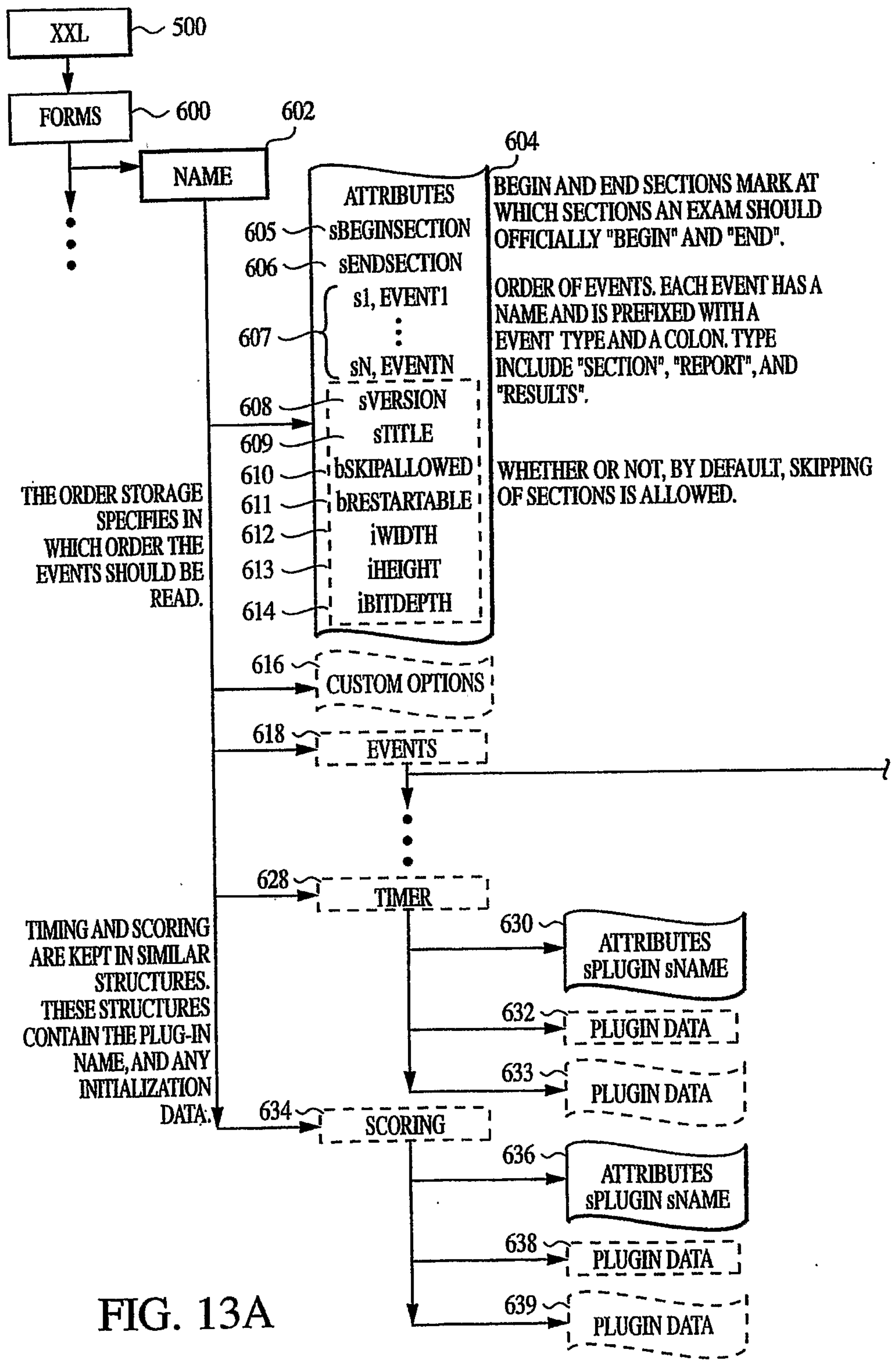

XXL
500
FORMS
600
602
NAME
604
ATTRIBUTES
605 sBEGINSECTION
606 sENDSECTION
607 s1, EVENT1 ⋮ sN, EVENTN
608 sVERSION
609 sTITLE
610 bSKIPALLOWED
611 bRESTARTABLE
612 iWIDTH
613 iHEIGHT
614 iBITDEPTH
BEGIN AND END SECTIONS MARK AT WHICH SECTIONS AN EXAM SHOULD OFFICIALLY "BEGIN" AND "END".
ORDER OF EVENTS. EACH EVENT HAS A NAME AND IS PREFIXED WITH A EVENT TYPE AND A COLON. TYPE INCLUDE "SECTION", "REPORT", AND "RESULTS".
WHETHER OR NOT, BY DEFAULT, SKIPPING OF SECTIONS IS ALLOWED.
THE ORDER STORAGE SPECIFIES IN WHICH ORDER THE EVENTS SHOULD BE READ.
616 CUSTOM OPTIONS
618 EVENTS
628 TIMER
630 ATTRIBUTES sPLUGIN sNAME
632 PLUGIN DATA
633 PLUGIN DATA
TIMING AND SCORING ARE KEPT IN SIMILAR STRUCTURES. THESE STRUCTURES CONTAIN THE PLUG-IN NAME, AND ANY INITIALIZATION DATA.
634 SCORING
636 ATTRIBUTES sPLUGIN sNAME
638 PLUGIN DATA
639 PLUGIN DATA

FIG. 13A

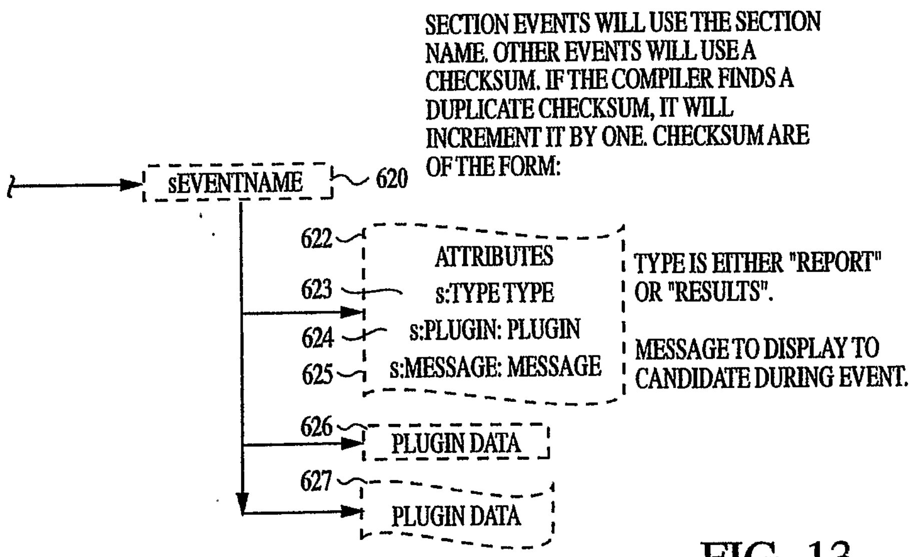

SECTION EVENTS WILL USE THE SECTION NAME. OTHER EVENTS WILL USE A CHECKSUM. IF THE COMPILER FINDS A DUPLICATE CHECKSUM, IT WILL INCREMENT IT BY ONE. CHECKSUM ARE OF THE FORM:
sEVENTNAME
620
622
ATTRIBUTES
623
s:TYPE TYPE
624
s:PLUGIN: PLUGIN
625
s:MESSAGE: MESSAGE
TYPE IS EITHER "REPORT" OR "RESULTS".
MESSAGE TO DISPLAY TO CANDIDATE DURING EVENT.
626
PLUGIN DATA
627
PLUGIN DATA

FIG. 13

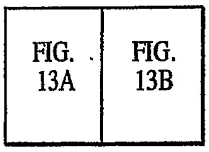


FIG. 13B

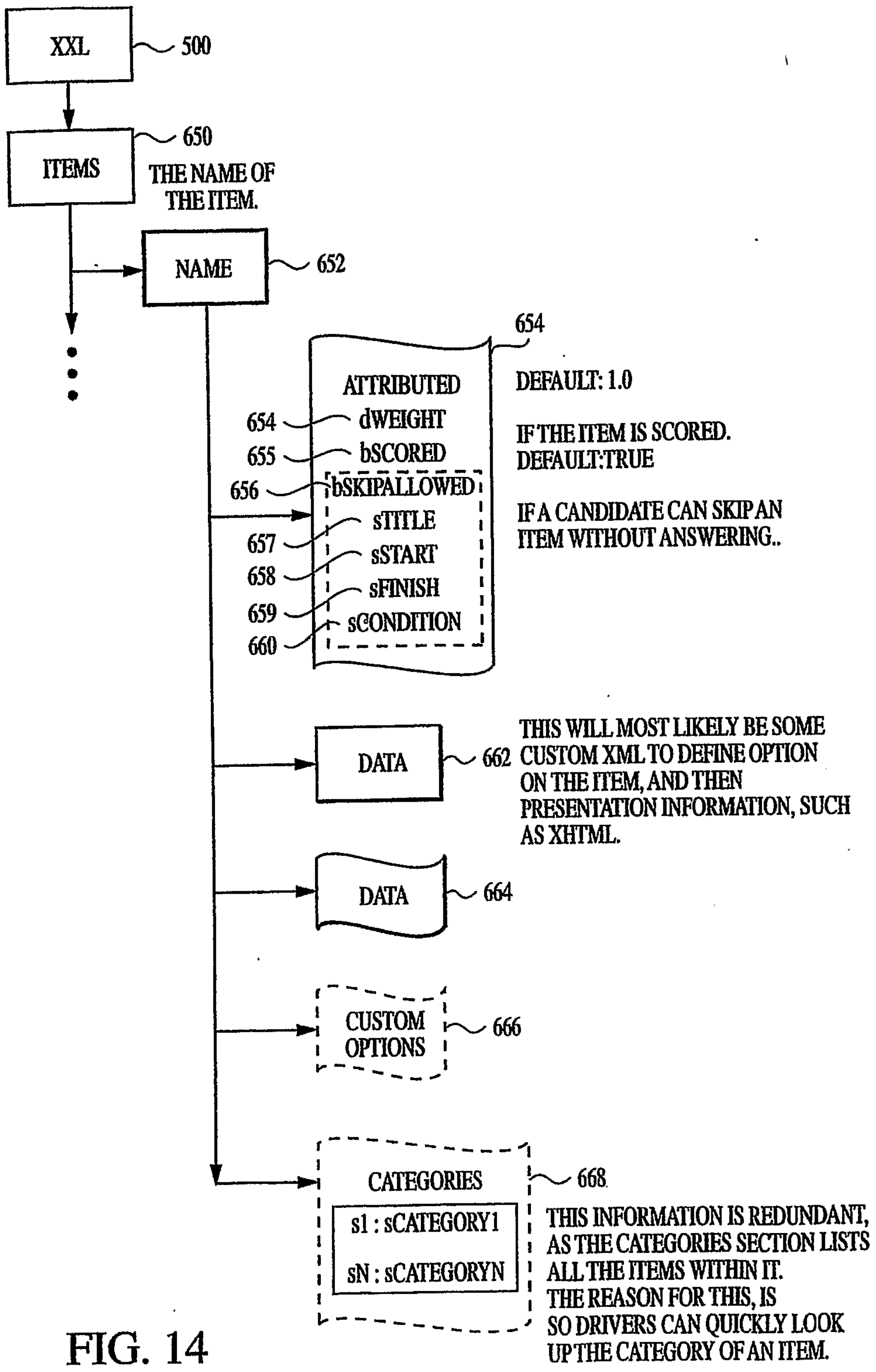

XXL
500
ITEMS
650
THE NAME OF THE ITEM.
NAME
652
654
ATTRIBUTED
654 dWEIGHT
655 bSCORED
656 bSKIPALLOWED
sTITLE
657 sSTART
658 sFINISH
659 sCONDITION
660
DEFAULT: 1.0
IF THE ITEM IS SCORED. DEFAULT:TRUE
IF A CANDIDATE CAN SKIP AN ITEM WITHOUT ANSWERING..
DATA
662
THIS WILL MOST LIKELY BE SOME CUSTOM XML TO DEFINE OPTION ON THE ITEM, AND THEN PRESENTATION INFORMATION, SUCH AS XHTML.
DATA
664
CUSTOM OPTIONS
666
CATEGORIES
668
s1 : sCATEGORY1
sN : sCATEGORYN
THIS INFORMATION IS REDUNDANT, AS THE CATEGORIES SECTION LISTS ALL THE ITEMS WITHIN IT. THE REASON FOR THIS, IS SO DRIVERS CAN QUICKLY LOOK UP THE CATEGORY OF AN ITEM.

FIG. 14

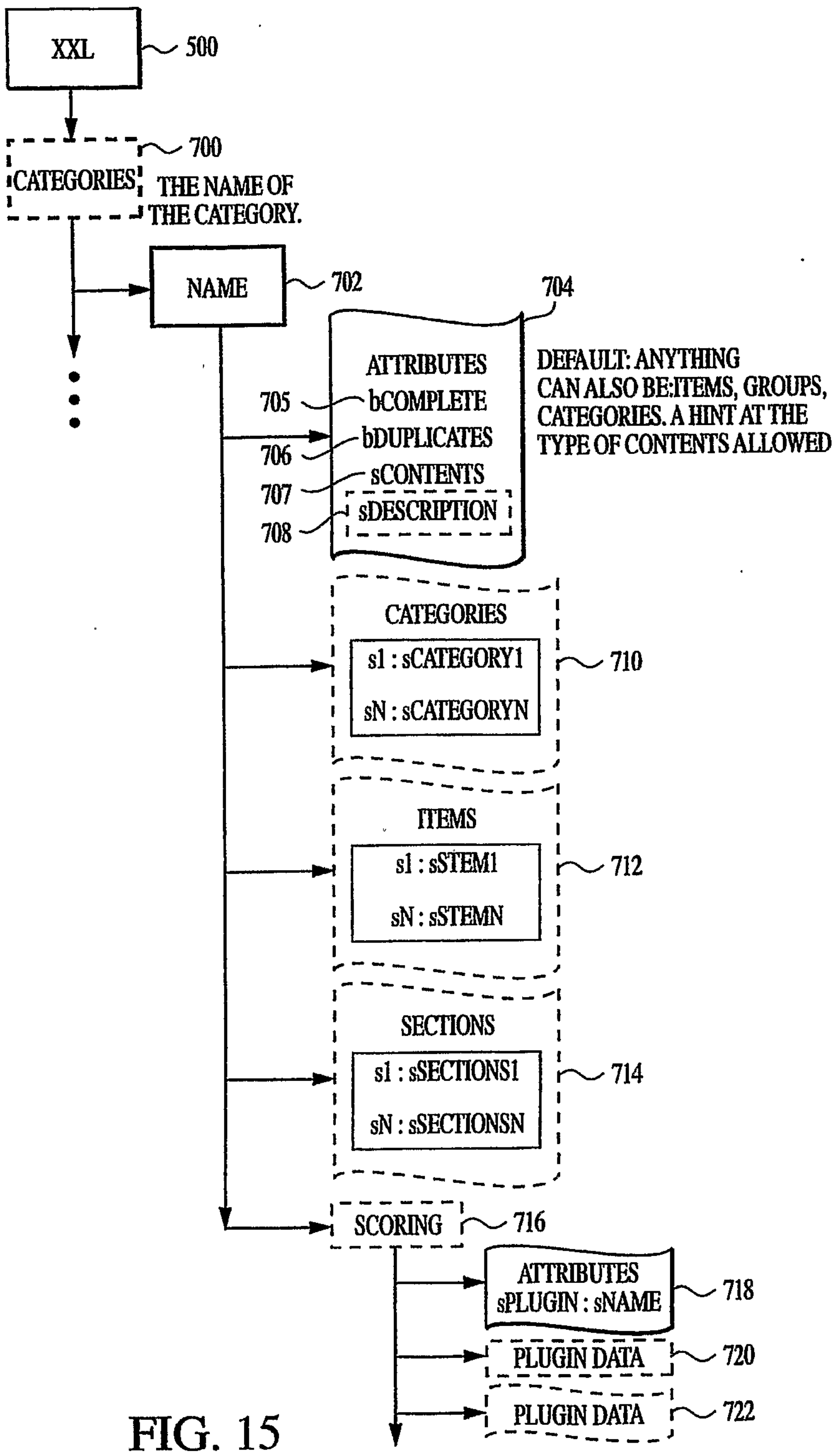

XXL
500
700
CATEGORIES
THE NAME OF THE CATEGORY.
NAME
702
704
ATTRIBUTES
705 bCOMPLETE
706 bDUPLICATES
707 sCONTENTS
708 sDESCRIPTION
DEFAULT: ANYTHING CAN ALSO BE:ITEMS, GROUPS, CATEGORIES. A HINT AT THE TYPE OF CONTENTS ALLOWED
CATEGORIES
s1 : sCATEGORY1
sN : sCATEGORYN
710
ITEMS
s1 : sSTEM1
sN : sSTEMN
712
SECTIONS
s1 : sSECTIONS1
sN : sSECTIONSN
714
SCORING
716
ATTRIBUTES
sPLUGIN : sNAME
718
PLUGIN DATA
720
PLUGIN DATA
722

FIG. 15

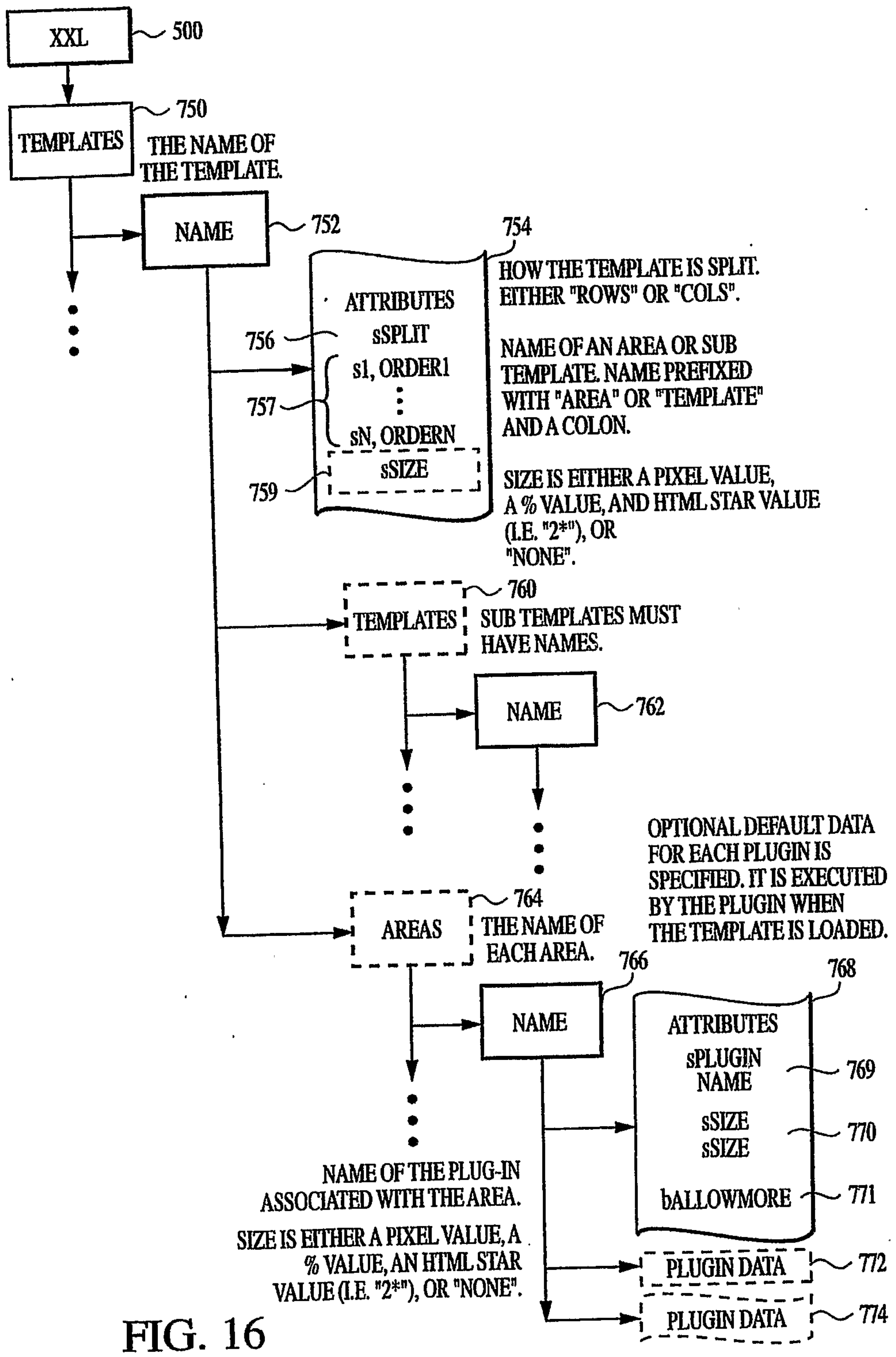

XXL
500
TEMPLATES
750
THE NAME OF THE TEMPLATE.
NAME
752
754
ATTRIBUTES
sSPLIT
s1, ORDER1
sN, ORDERN
sSIZE
756
757
759
HOW THE TEMPLATE IS SPLIT. EITHER "ROWS" OR "COLS".
NAME OF AN AREA OR SUB TEMPLATE. NAME PREFIXED WITH "AREA" OR "TEMPLATE" AND A COLON.
SIZE IS EITHER A PIXEL VALUE, A % VALUE, AND HTML STAR VALUE (I.E. "2*"), OR "NONE".
TEMPLATES
760
SUB TEMPLATES MUST HAVE NAMES.
NAME
762
OPTIONAL DEFAULT DATA FOR EACH PLUGIN IS SPECIFIED. IT IS EXECUTED BY THE PLUGIN WHEN THE TEMPLATE IS LOADED.
AREAS
764
THE NAME OF EACH AREA.
NAME
766
768
ATTRIBUTES
sPLUGIN NAME
769
sSIZE sSIZE
770
bALLOWMORE
771
NAME OF THE PLUG-IN ASSOCIATED WITH THE AREA.
SIZE IS EITHER A PIXEL VALUE, A % VALUE, AN HTML STAR VALUE (I.E. "2*"), OR "NONE".
PLUGIN DATA
772
PLUGIN DATA
774

FIG. 16

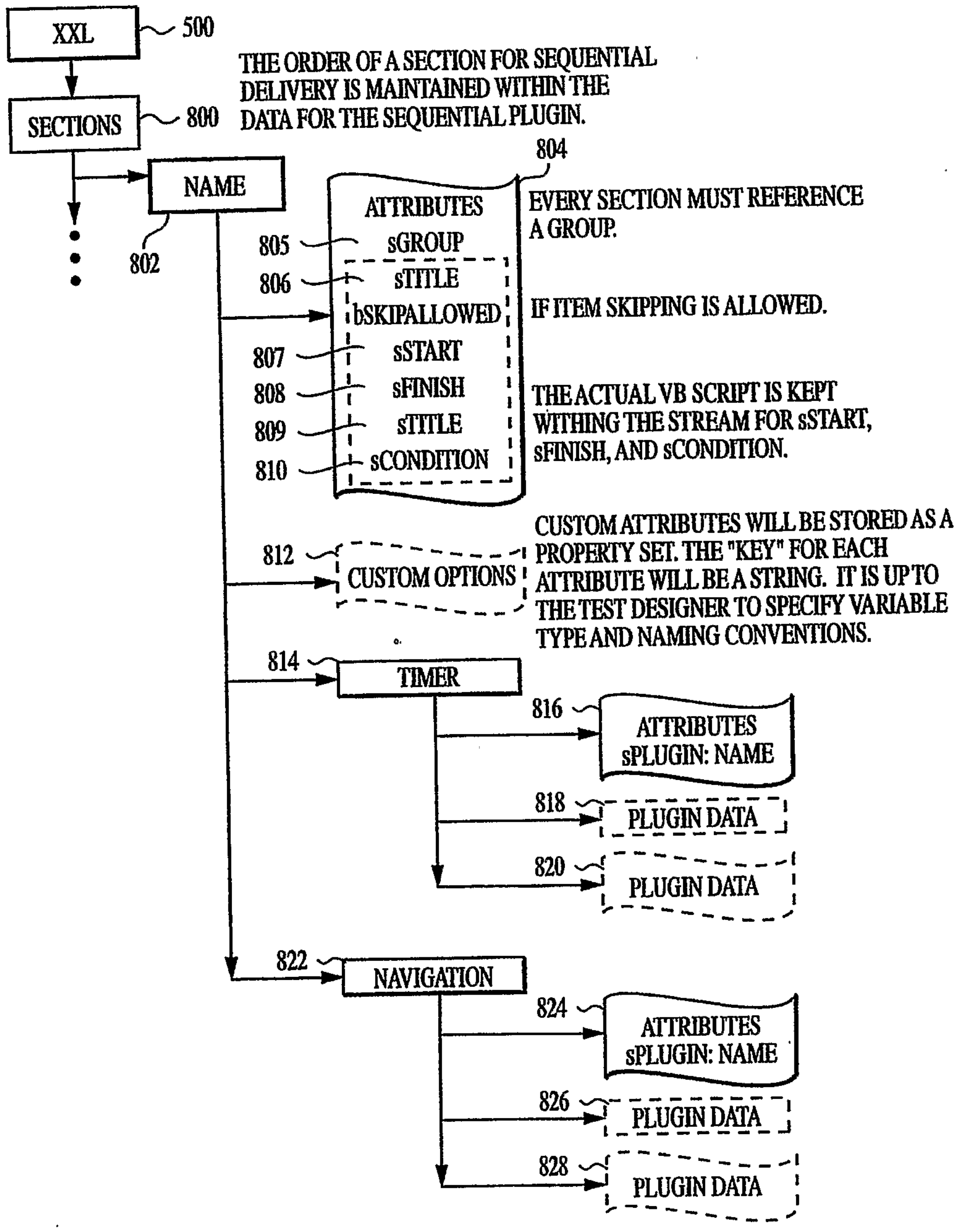

XXL
500
SECTIONS
800
THE ORDER OF A SECTION FOR SEQUENTIAL DELIVERY IS MAINTAINED WITHIN THE DATA FOR THE SEQUENTIAL PLUGIN.
NAME
802
804
ATTRIBUTES
805 sGROUP
806 sTITLE
bSKIPALLOWED
807 sSTART
808 sFINISH
809 sTITLE
810 sCONDITION
EVERY SECTION MUST REFERENCE A GROUP.
IF ITEM SKIPPING IS ALLOWED.
THE ACTUAL VB SCRIPT IS KEPT WITHING THE STREAM FOR sSTART, sFINISH, AND sCONDITION.
812 CUSTOM OPTIONS
CUSTOM ATTRIBUTES WILL BE STORED AS A PROPERTY SET. THE "KEY" FOR EACH ATTRIBUTE WILL BE A STRING. IT IS UP TO THE TEST DESIGNER TO SPECIFY VARIABLE TYPE AND NAMING CONVENTIONS.
814 TIMER
816 ATTRIBUTES sPLUGIN: NAME
818 PLUGIN DATA
820 PLUGIN DATA
822 NAVIGATION
824 ATTRIBUTES sPLUGIN: NAME
826 PLUGIN DATA
828 PLUGIN DATA

FIG. 17

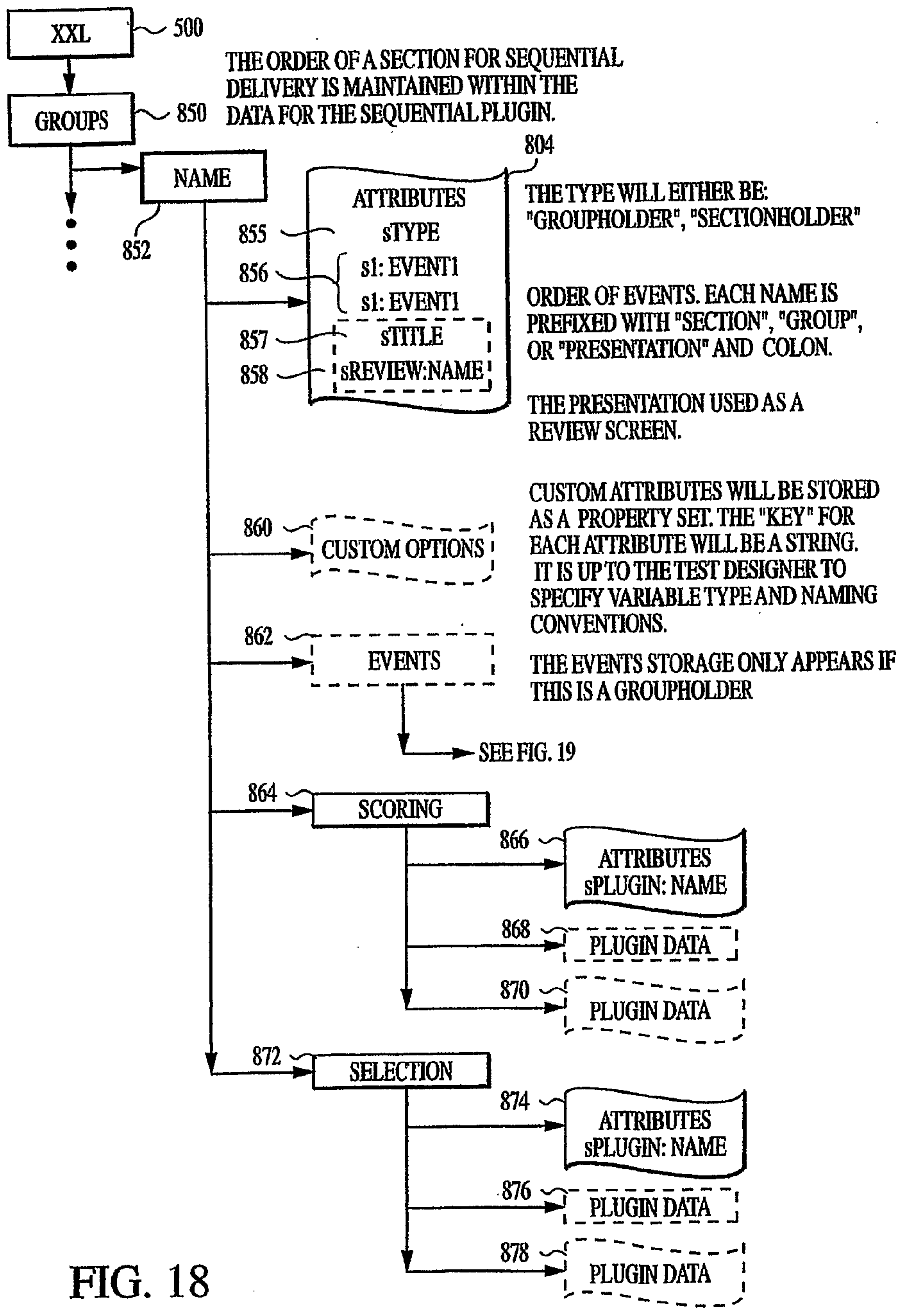

XXL
500
GROUPS
850
THE ORDER OF A SECTION FOR SEQUENTIAL DELIVERY IS MAINTAINED WITHIN THE DATA FOR THE SEQUENTIAL PLUGIN.
NAME
852
804
ATTRIBUTES
855
sTYPE
856
s1: EVENT1
s1: EVENT1
857
sTITLE
858
sREVIEW:NAME
THE TYPE WILL EITHER BE: "GROUPHOLDER", "SECTIONHOLDER"
ORDER OF EVENTS. EACH NAME IS PREFIXED WITH "SECTION", "GROUP", OR "PRESENTATION" AND COLON.
THE PRESENTATION USED AS A REVIEW SCREEN.
860
CUSTOM OPTIONS
CUSTOM ATTRIBUTES WILL BE STORED AS A PROPERTY SET. THE "KEY" FOR EACH ATTRIBUTE WILL BE A STRING. IT IS UP TO THE TEST DESIGNER TO SPECIFY VARIABLE TYPE AND NAMING CONVENTIONS.
862
EVENTS
THE EVENTS STORAGE ONLY APPEARS IF THIS IS A GROUPHOLDER
SEE FIG. 19
864
SCORING
866
ATTRIBUTES sPLUGIN: NAME
868
PLUGIN DATA
870
PLUGIN DATA
872
SELECTION
874
ATTRIBUTES sPLUGIN: NAME
876
PLUGIN DATA
878
PLUGIN DATA

FIG. 18

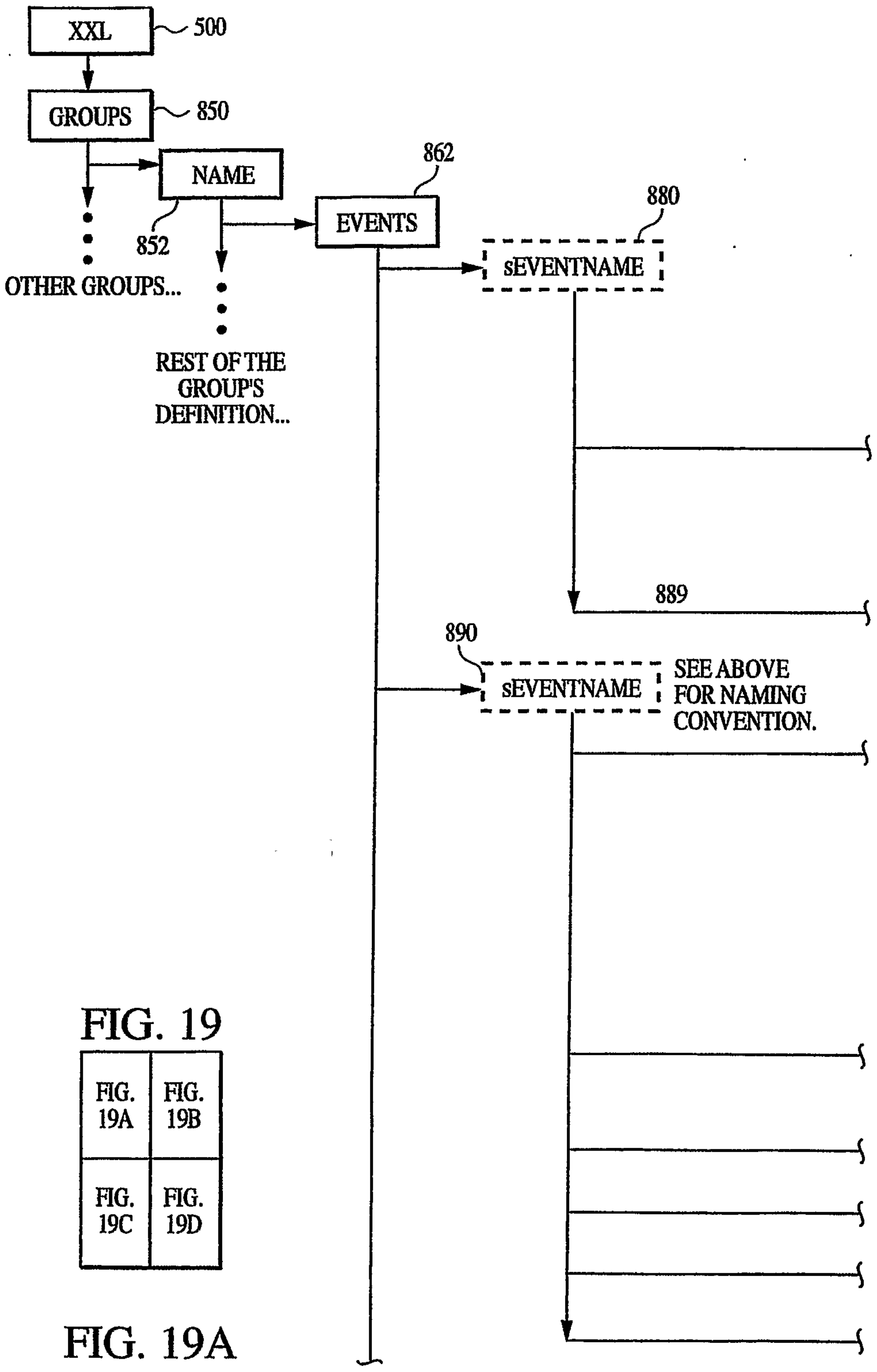
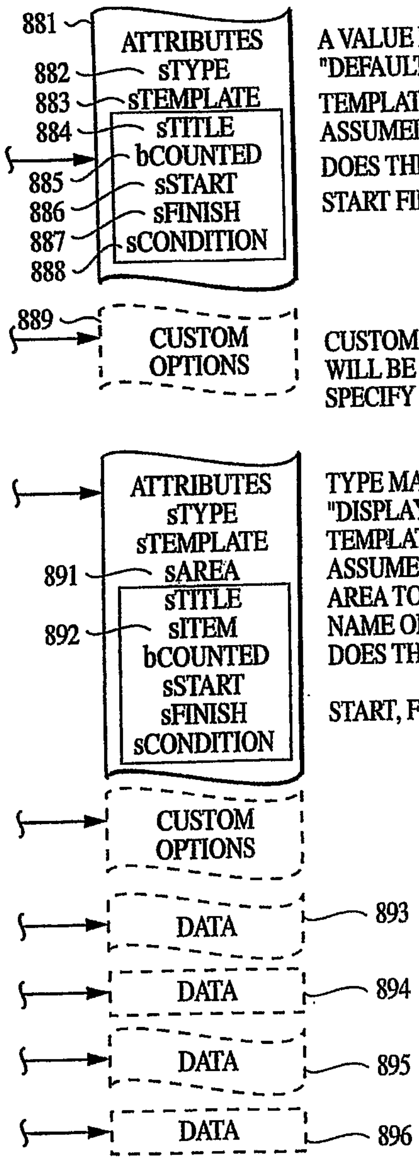

XXL
500
GROUPS
850
NAME
852
OTHER GROUPS...
REST OF THE GROUP'S DEFINITION...
EVENTS
862
sEVENTNAME
880
889
890
sEVENTNAME
SEE ABOVE FOR NAMING CONVENTION.
FIG. 19
FIG. 19A
FIG. 19B
FIG. 19C
FIG. 19D

FIG. 19A

EACH PRESENTATION STORAGE WILL HAVE A NAME FROM THE XXL OR A CHECKSUM. IF NO NAME IS SPECIFIED A CHECKSUM IS CREATED BASED ON THE CONTENTS. IF THE COMPILER FINDS A DUPLICATE CHECKSUM, IT WILL INCREMENT IT BY ONE. CHECKSUM ARE OF THE FORM



A VALUE FOR THE PRESENTATION TYPE WHICH MUST BE "DEFAULT".

TEMPLATE FOR THIS PRESENTATION. ALL AREAS ARE ASSUMED TO HAVE DATA SENT TO THEM.

DOES THIS APPEAR IN A "X OF Y" DISPLAY?

START FINISH AND CONDITIONAL SCRIPTS.

CUSTOM ATTRIBUTES. THE "KEY" FOR EACH ATTRIBUTE WILL BE A STRING. IT IS UP TO THE TEST DESIGNER TO SPECIFY VARIABLE TYPE AND NAMING CONVENTIONS.

TYPE MAY BE "ITEM" FOR A SINGLE AREA ITEM, OR "DISPLAY" FOR A SINGLE AREA DISPLAY.

TEMPLATE FOR THIS PRESENTATION. ALL AREAS ARE ASSUMED TO HAVE DATA SENT TO THEM.

AREA TO RENDER THE PRESENTATION'S CONTENT.

NAME OF THE ASSOCIATED ITEM IF OF TYPE "ITEM".

DOES THIS APPEAR IN A "X OF Y" DISPLAY?

START, FINISH, AND CONDITIONAL SCRIPTS.

FIG. 19B

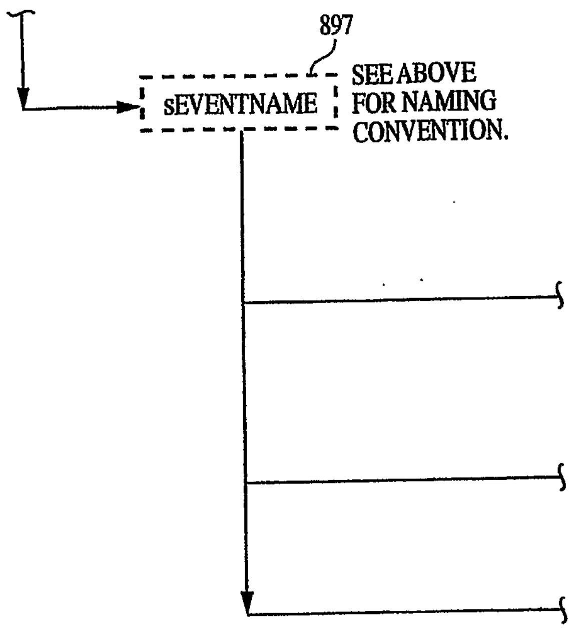
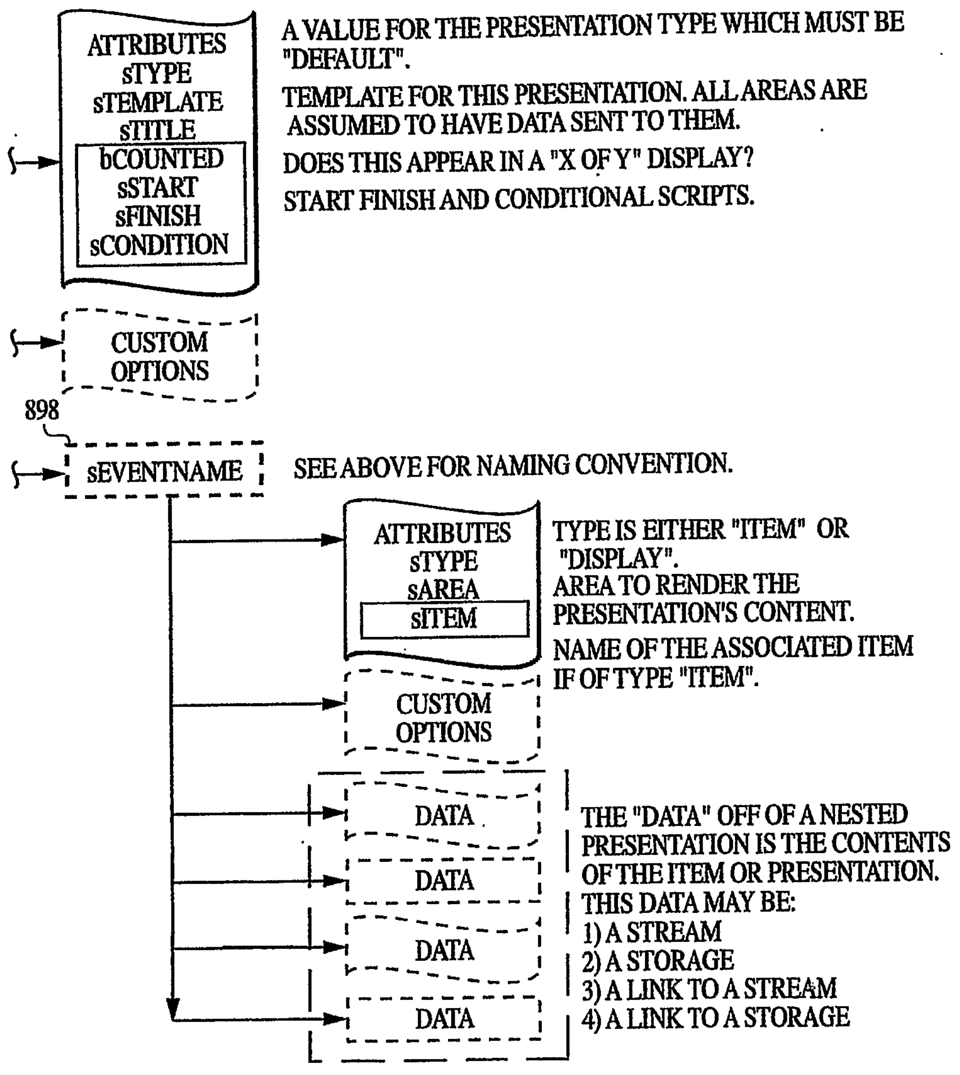


FIG. 19C



ATTRIBUTES
sTYPE
sTEMPLATE
sTITLE
bCOUNTED
sSTART
sFINISH
sCONDITION
A VALUE FOR THE PRESENTATION TYPE WHICH MUST BE "DEFAULT".
TEMPLATE FOR THIS PRESENTATION. ALL AREAS ARE ASSUMED TO HAVE DATA SENT TO THEM.
DOES THIS APPEAR IN A "X OF Y" DISPLAY?
START FINISH AND CONDITIONAL SCRIPTS.
CUSTOM OPTIONS
898
sEVENTNAME
SEE ABOVE FOR NAMING CONVENTION.
ATTRIBUTES
sTYPE
sAREA
sITEM
TYPE IS EITHER "ITEM" OR "DISPLAY".
AREA TO RENDER THE PRESENTATION'S CONTENT.
NAME OF THE ASSOCIATED ITEM IF OF TYPE "ITEM".
CUSTOM OPTIONS
DATA
DATA
DATA
DATA
THE "DATA" OFF OF A NESTED PRESENTATION IS THE CONTENTS OF THE ITEM OR PRESENTATION. THIS DATA MAY BE:
1) A STREAM
2) A STORAGE
3) A LINK TO A STREAM
4) A LINK TO A STORAGE

FIG. 19D

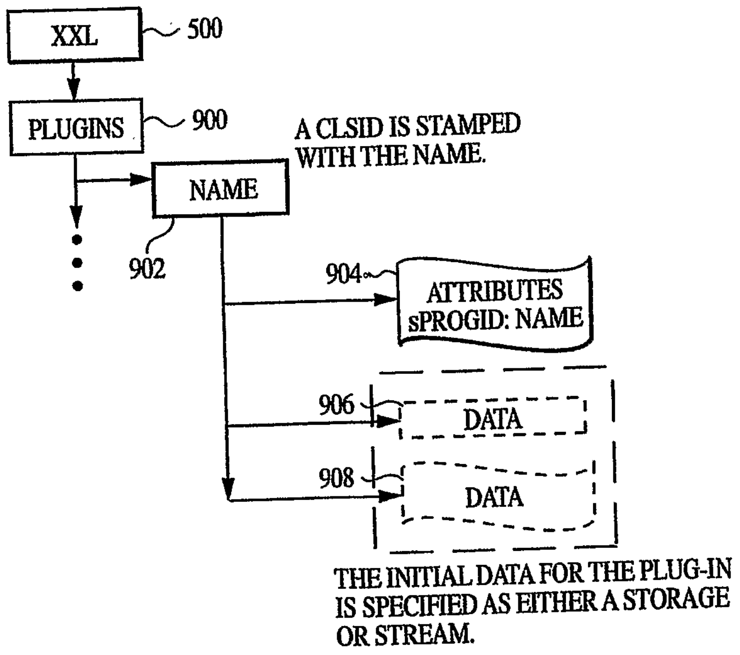

XXL
500
PLUGINS
900
A CLSID IS STAMPED WITH THE NAME.
NAME
902
904
ATTRIBUTES sPROGID: NAME
906
DATA
908
DATA
THE INITIAL DATA FOR THE PLUG-IN IS SPECIFIED AS EITHER A STORAGE OR STREAM.

FIG. 20

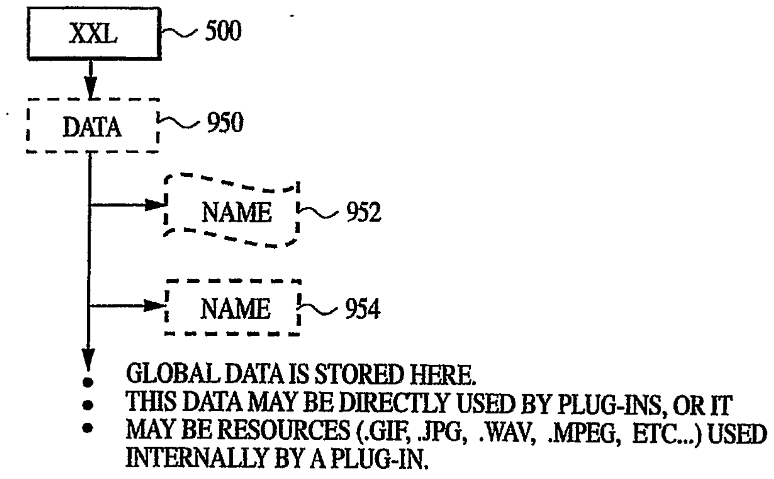

XXL
500
DATA
950
NAME
952
NAME
954
GLOBAL DATA IS STORED HERE.
THIS DATA MAY BE DIRECTLY USED BY PLUG-INS, OR IT MAY BE RESOURCES (.GIF, .JPG, .WAV, .MPEG, ETC...) USED INTERNALLY BY A PLUG-IN.

FIG. 21

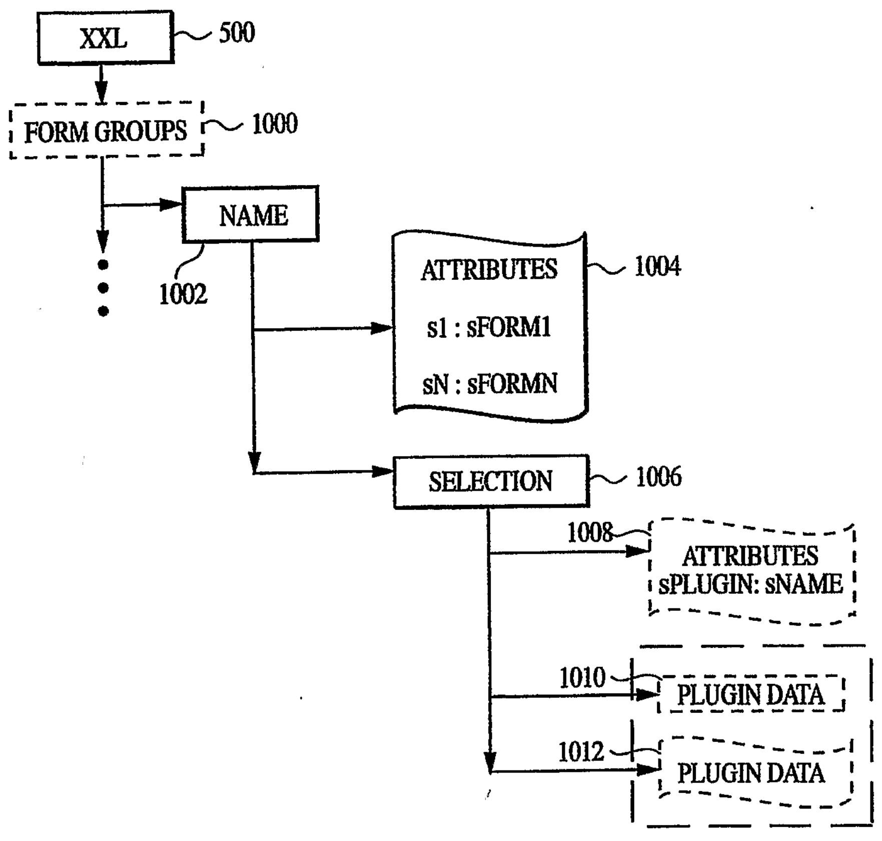

XXL
500
FORM GROUPS
1000
NAME
1002
ATTRIBUTES
s1 : sFORM1
sN : sFORMN
1004
SELECTION
1006
1008
ATTRIBUTES
sPLUGIN: sNAME
1010
PLUGIN DATA
1012
PLUGIN DATA

FIG. 22

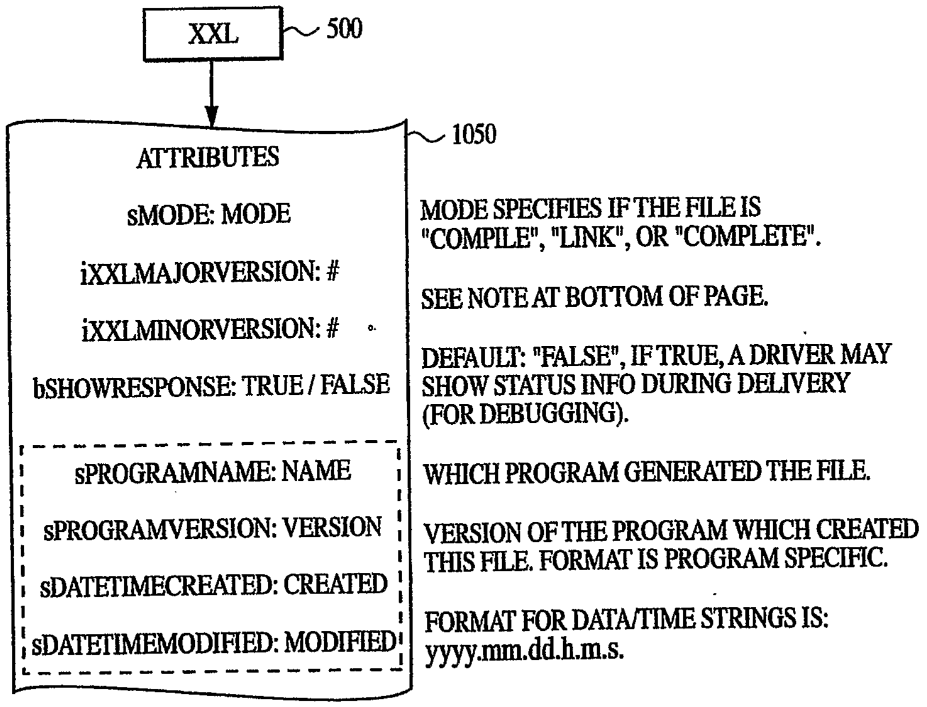

XXL
500
1050
ATTRIBUTES
sMODE: MODE
iXXLMAJORVERSION: #
iXXLMINORVERSION: #
bSHOWRESPONSE: TRUE / FALSE
sPROGRAMNAME: NAME
sPROGRAMVERSION: VERSION
sDATETIMECREATED: CREATED
sDATETIMEMODIFIED: MODIFIED
MODE SPECIFIES IF THE FILE IS "COMPILE", "LINK", OR "COMPLETE".
SEE NOTE AT BOTTOM OF PAGE.
DEFAULT: "FALSE", IF TRUE, A DRIVER MAY SHOW STATUS INFO DURING DELIVERY (FOR DEBUGGING).
WHICH PROGRAM GENERATED THE FILE.
VERSION OF THE PROGRAM WHICH CREATED THIS FILE. FORMAT IS PROGRAM SPECIFIC.
FORMAT FOR DATA/TIME STRINGS IS: yyyy.mm.dd.h.m.s.

FIG. 23

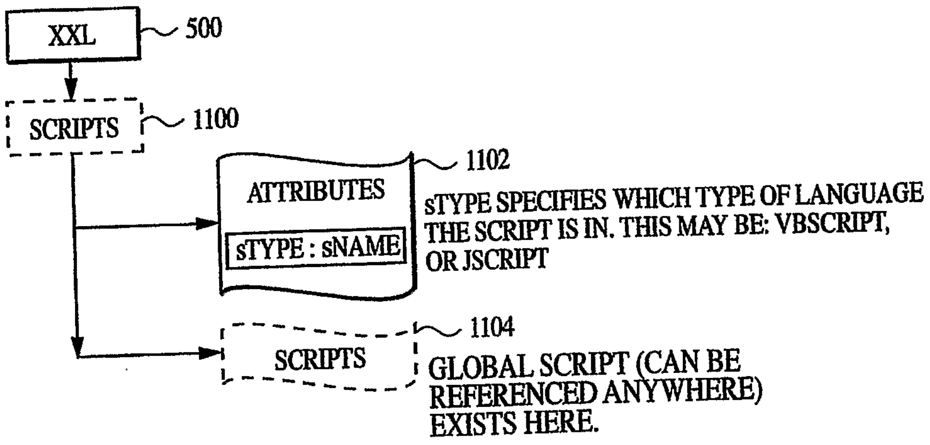
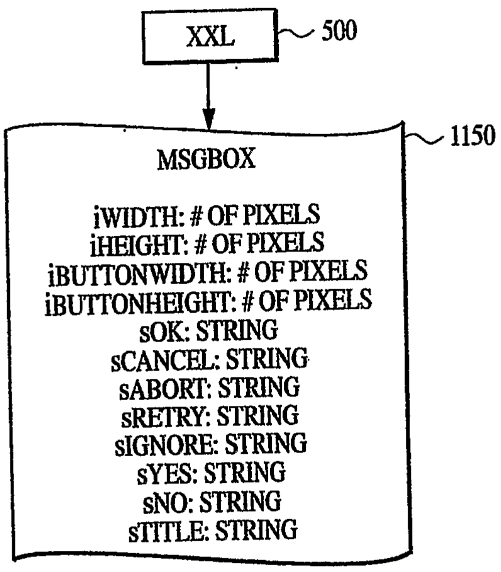

XXL
500
SCRIPTS
1100
ATTRIBUTES
sTYPE : sNAME
1102
sTYPE SPECIFIES WHICH TYPE OF LANGUAGE THE SCRIPT IS IN. THIS MAY BE: VBSCRIPT, OR JSCRIPT
SCRIPTS
1104
GLOBAL SCRIPT (CAN BE REFERENCED ANYWHERE) EXISTS HERE.

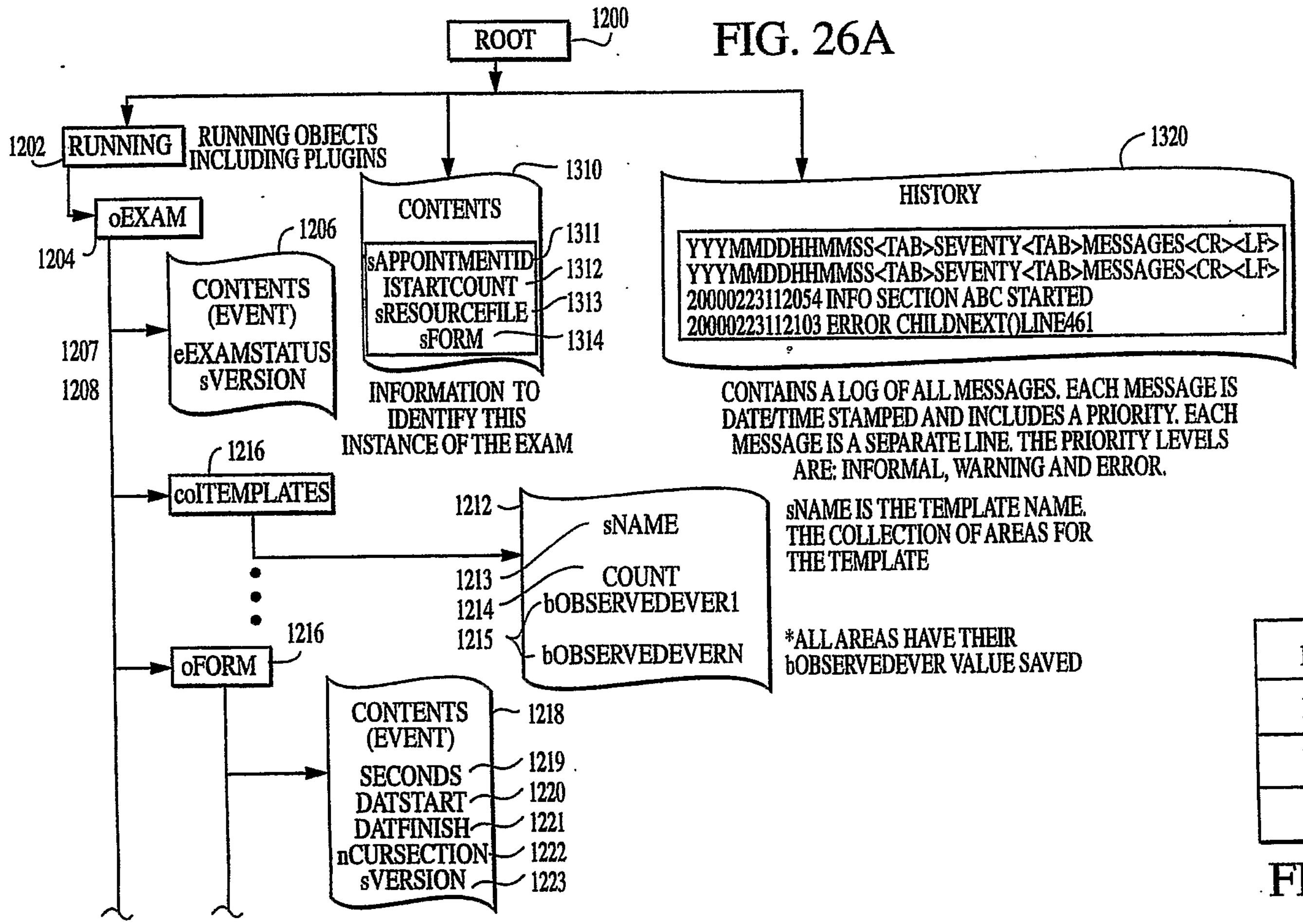
FIG. 24



MESSAGE BOXES MAY BE TRIGGERED BY PLUG-INS IN THE EXAM. THE LOOK AND TEST OF THE BUTTONS/BOX ARE CONTROLLED HERE.

FIG. 25

FIG. 26A


ROOT
1200
1202
RUNNING
RUNNING OBJECTS INCLUDING PLUGINS
oEXAM
1204
1206
CONTENTS (EVENT)
eEXAMSTATUS
sVERSION
1207
1208
1310
CONTENTS
sAPPOINTMENTID
ISTARTCOUNT
sRESOURCEFILE
sFORM
1311
1312
1313
1314
INFORMATION TO IDENTIFY THIS INSTANCE OF THE EXAM
1320
HISTORY
YYYMMDDHHMMSS<TAB>SEVENTY<TAB>MESSAGES<CR><LF>
YYYMMDDHHMMSS<TAB>SEVENTY<TAB>MESSAGES<CR><LF>
20000223112054 INFO SECTION ABC STARTED
20000223112103 ERROR CHILDNEXT()LINE461
CONTAINS A LOG OF ALL MESSAGES. EACH MESSAGE IS DATE/TIME STAMPED AND INCLUDES A PRIORITY. EACH MESSAGE IS A SEPARATE LINE. THE PRIORITY LEVELS ARE: INFORMAL, WARNING AND ERROR.
1216
colTEMPLATES
1212
sNAME
COUNT
bOBSERVEDEVER1
bOBSERVEDEVERN
1213
1214
1215
sNAME IS THE TEMPLATE NAME. THE COLLECTION OF AREAS FOR THE TEMPLATE
*ALL AREAS HAVE THEIR bOBSERVEDEVER VALUE SAVED
1216
oFORM
CONTENTS (EVENT)
SECONDS
DATSTART
DATFINISH
nCURSECTION
sVERSION
1218
1219
1220
1221
1222
1223

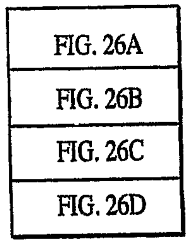


FIG. 26

FIG. 26B

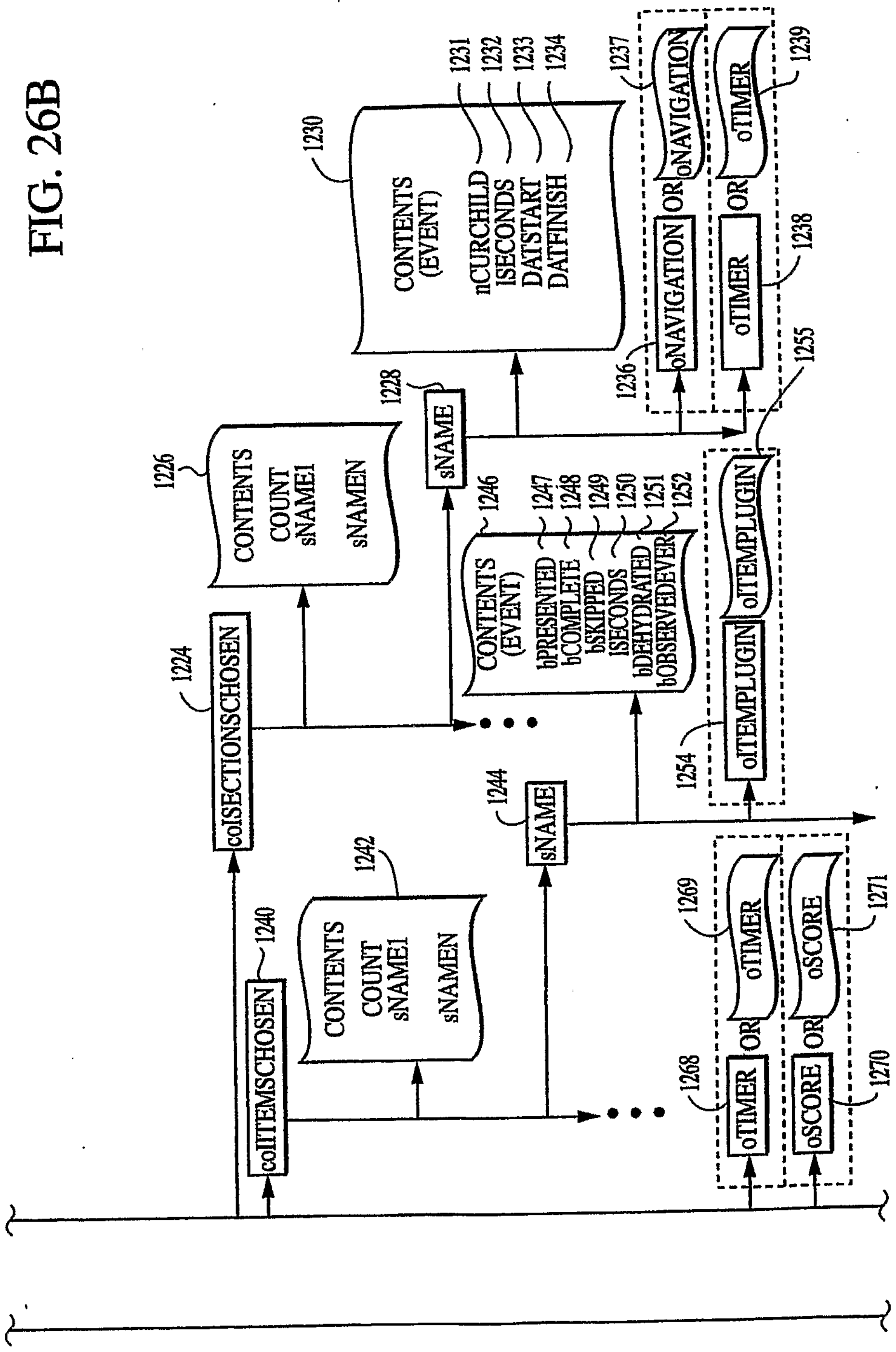
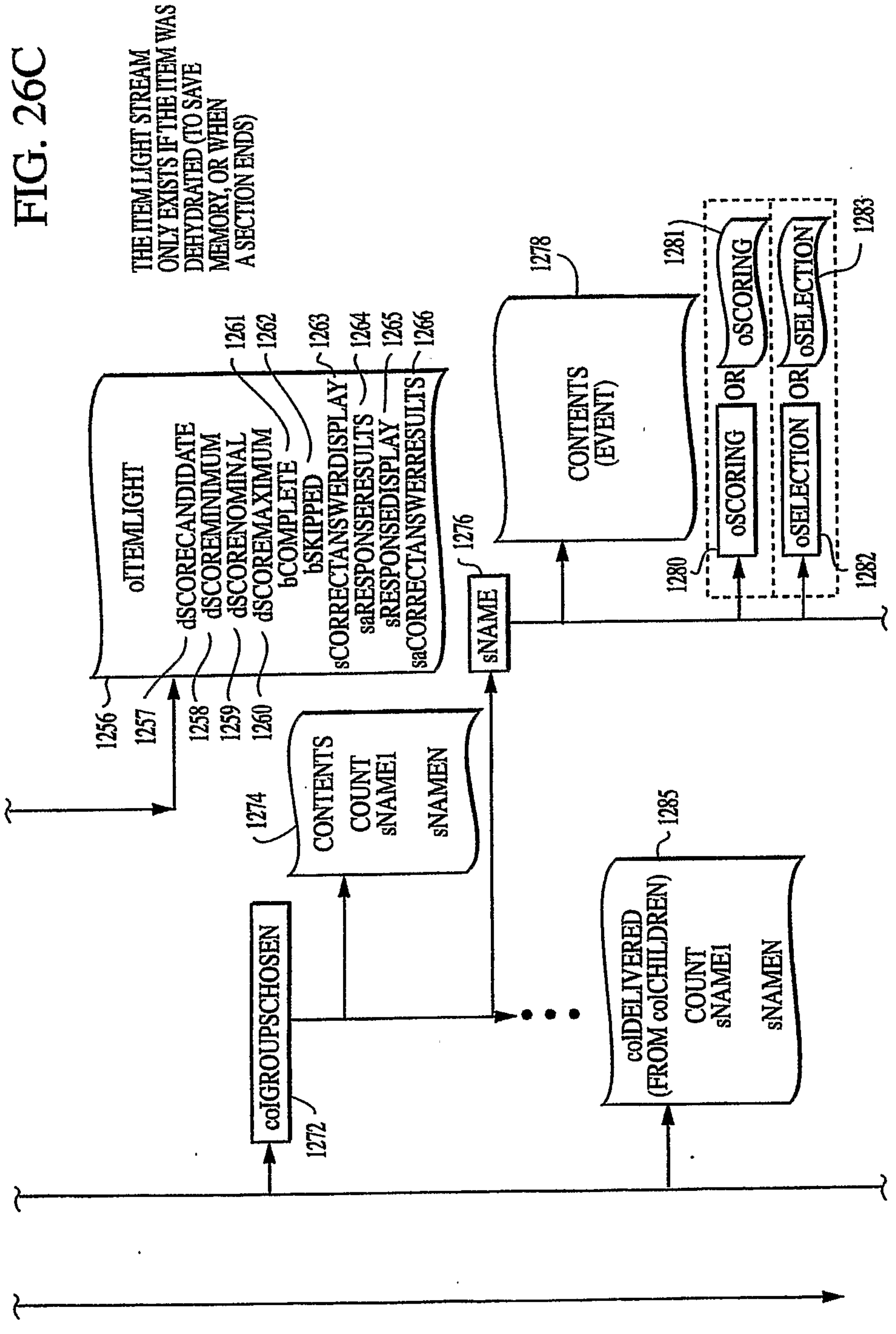
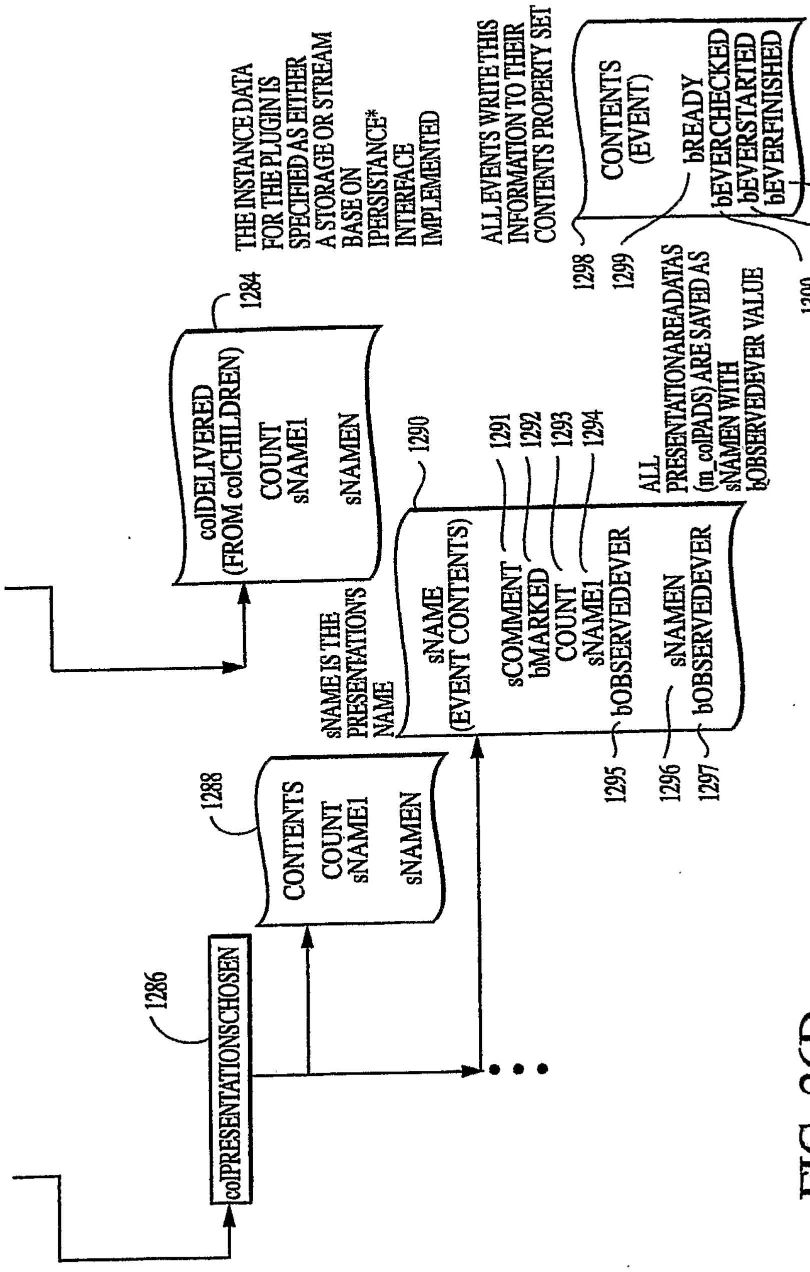

coISECTIONSCHOSEN
1224
CONTENTS
COUNT
sNAME1
sNAMEN
1226
sNAME
1228
CONTENTS
(EVENT)
nCURCHILD
lSECONDS
DATSTART
DATFINISH
1230
1231
1232
1233
1234
oNAVIGATION
OR
oNAVIGATION
1236
1237
oTIMER
OR
oTIMER
1238
1239
1255
CONTENTS
(EVENT)
bPRESENTED
bCOMPLETE
bSKIPPED
lSECONDS
bDEHYDRATED
bOBSERVEDEVER
1246
1247
1248
1249
1250
1251
1252
oITEMPLUGIN
oITEMPLUGIN
1254
coIITEMSCHOSEN
1240
CONTENTS
COUNT
sNAME1
sNAMEN
1242
sNAME
1244
oTIMER
OR
oTIMER
1268
1269
oSCORE
OR
oSCORE
1270
1271

FIG. 26C


THE ITEM LIGHT STREAM ONLY EXISTS IF THE ITEM WAS DEHYDRATED (TO SAVE MEMORY, OR WHEN A SECTION ENDS)
oITEMLIGHT
dSCORECANDIDATE
dSCOREMINIMUM
dSCORENOMINAL
dSCOREMAXIMUM
bCOMPLETE
bSKIPPED
sCORRECTANSWERDISPLAY
saRESPONSERESULTS
sRESPONSEDISPLAY
saCORRECTANSWERRESULTS
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
colGROUPSCHOSEN
1272
CONTENTS
COUNT
sNAME1
sNAMEN
1274
sNAME
1276
CONTENTS (EVENT)
1278
oSCORING
OR
oSCORING
1280
1281
oSELECTION
OR
oSELECTION
1282
1283
colDELIVERED (FROM colCHILDREN)
COUNT
sNAME1
sNAMEN
1285



colPRESENTATIONSCHOSEN
1286
CONTENTS
COUNT
sNAME1
sNAMEN
1288
colDELIVERED
(FROM colCHILDREN)
COUNT
sNAME1
sNAMEN
1284
THE INSTANCE DATA FOR THE PLUGIN IS SPECIFIED AS EITHER A STORAGE OR STREAM BASE ON IPERSISTANCE* INTERFACE IMPLEMENTED
sNAME IS THE PRESENTATION'S NAME
sNAME
(EVENT CONTENTS)
sCOMMENT
bMARKED
COUNT
sNAME1
bOBSERVEDEVER
sNAMEN
bOBSERVEDEVER
1290
1291
1292
1293
1294
1295
1296
1297
ALL PRESENTATIONAREADATAS (m_colPADS) ARE SAVED AS sNAMEN WITH bOBSERVEDEVER VALUE
ALL EVENTS WRITE THIS INFORMATION TO THEIR CONTENTS PROPERTY SET
CONTENTS
(EVENT)
bREADY
bEVERCHECKED
bEVERSTARTED
bEVERFINISHED
1298
1299
1300
1301
1302

FIG. 26D

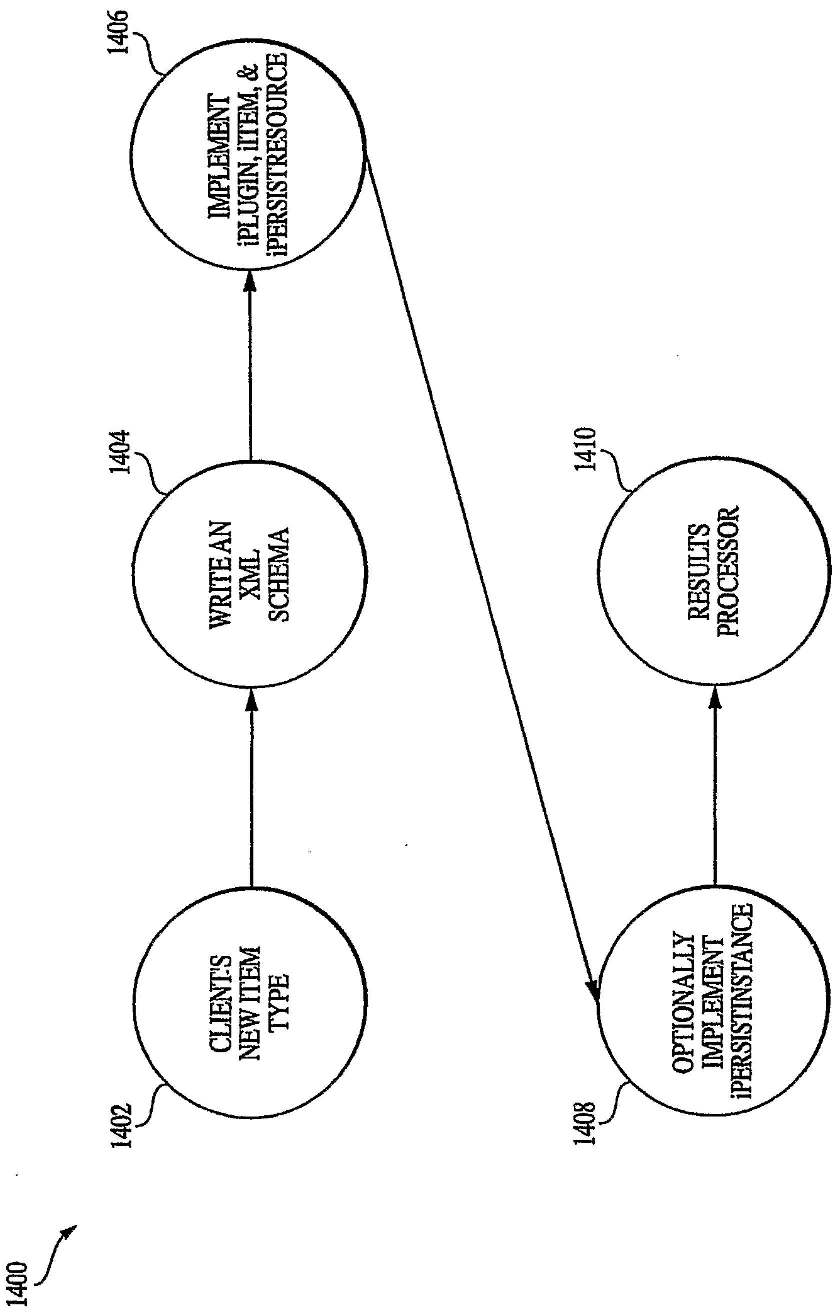

1400
1402
CLIENT'S NEW ITEM TYPE
1404
WRITE AN XML SCHEMA
1406
IMPLEMENT iPLUGIN, iITEM, & iPERSISTRESOURCE
1408
OPTIONALLY IMPLEMENT iPERSISTINSTANCE
1410
RESULTS PROCESSOR

FIG. 27

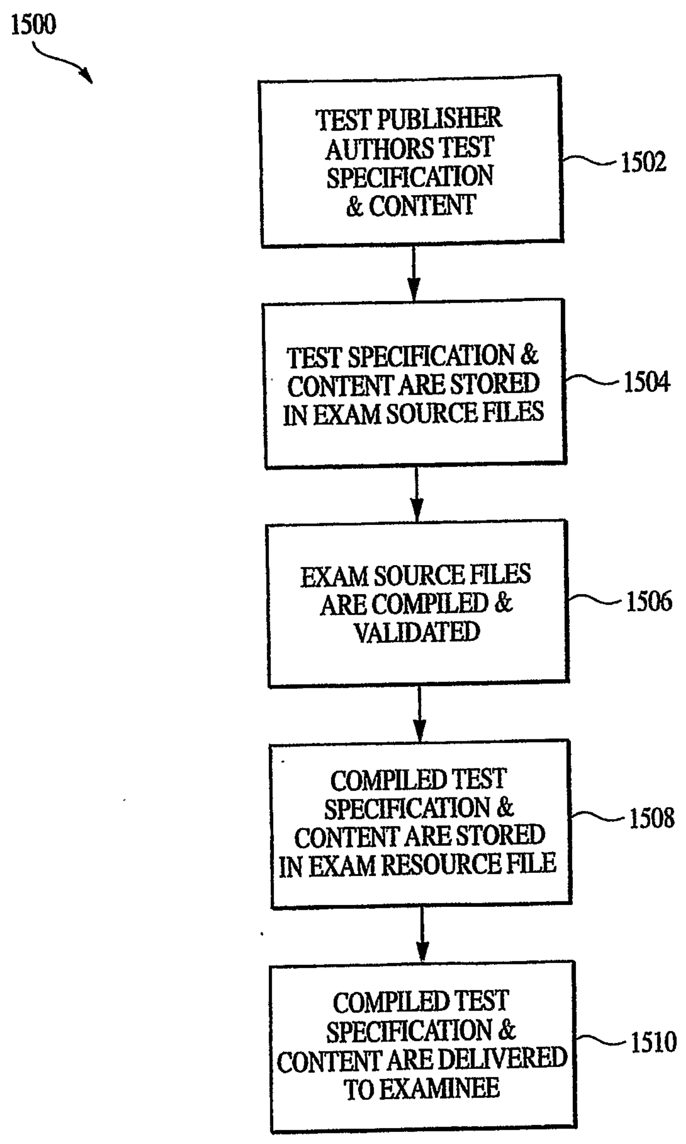

1500
TEST PUBLISHER AUTHORS TEST SPECIFICATION & CONTENT
1502
TEST SPECIFICATION & CONTENT ARE STORED IN EXAM SOURCE FILES
1504
EXAM SOURCE FILES ARE COMPILED & VALIDATED
1506
COMPILED TEST SPECIFICATION & CONTENT ARE STORED IN EXAM RESOURCE FILE
1508
COMPILED TEST SPECIFICATION & CONTENT ARE DELIVERED TO EXAMINEE
1510

FIG. 28

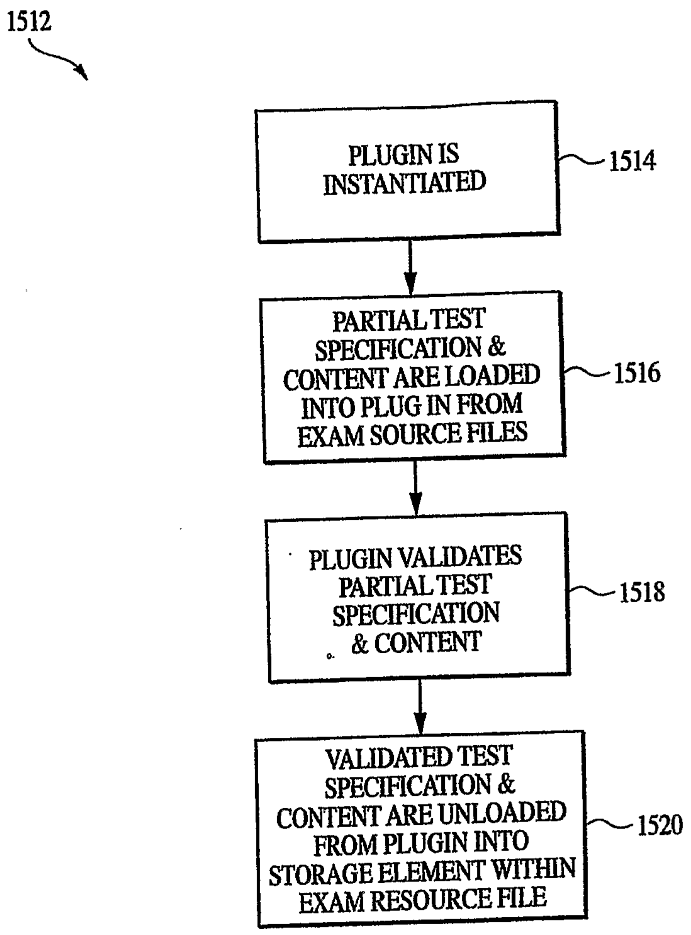

1512
PLUGIN IS INSTANTIATED
1514
PARTIAL TEST SPECIFICATION & CONTENT ARE LOADED INTO PLUG IN FROM EXAM SOURCE FILES
1516
PLUGIN VALIDATES PARTIAL TEST SPECIFICATION & CONTENT
1518
VALIDATED TEST SPECIFICATION & CONTENT ARE UNLOADED FROM PLUGIN INTO STORAGE ELEMENT WITHIN EXAM RESOURCE FILE
1520

FIG. 29

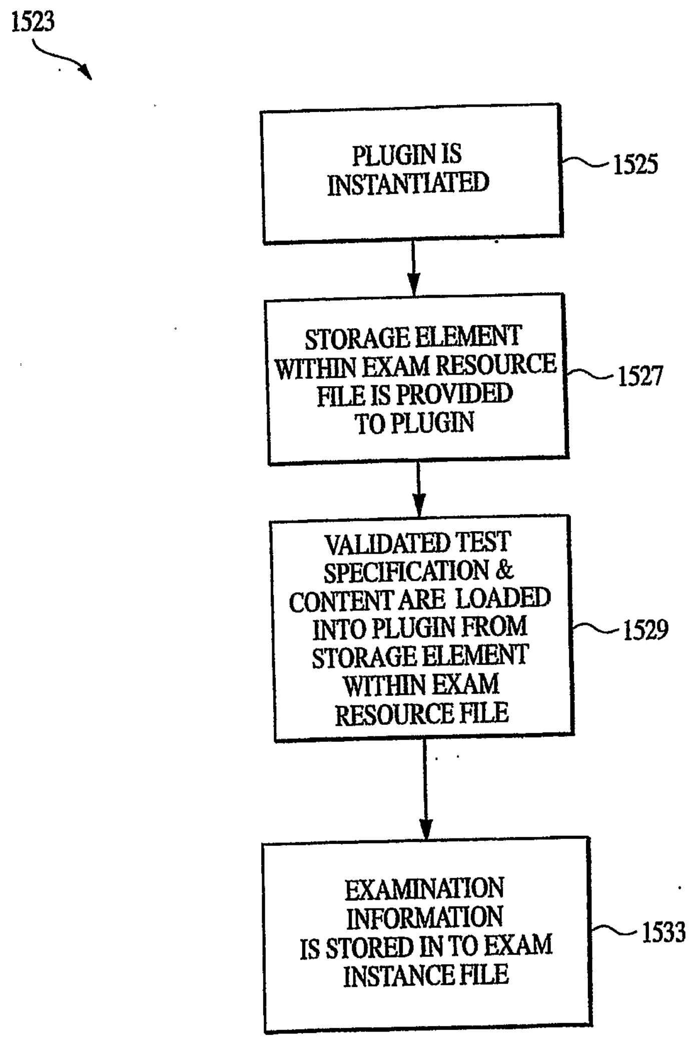

1523
PLUGIN IS INSTANTIATED
1525
STORAGE ELEMENT WITHIN EXAM RESOURCE FILE IS PROVIDED TO PLUGIN
1527
VALIDATED TEST SPECIFICATION & CONTENT ARE LOADED INTO PLUGIN FROM STORAGE ELEMENT WITHIN EXAM RESOURCE FILE
1529
EXAMINATION INFORMATION IS STORED IN TO EXAM INSTANCE FILE
1533

FIG. 30

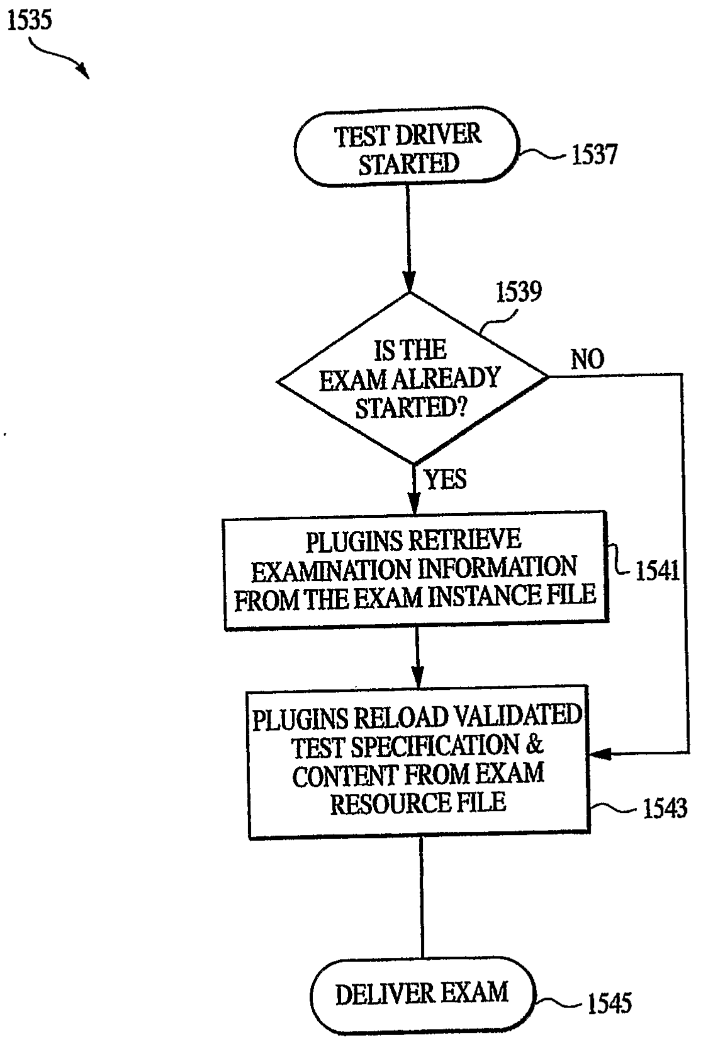

1535
TEST DRIVER STARTED
1537
1539
IS THE EXAM ALREADY STARTED?
NO
YES
PLUGINS RETRIEVE EXAMINATION INFORMATION FROM THE EXAM INSTANCE FILE
1541
PLUGINS RELOAD VALIDATED TEST SPECIFICATION & CONTENT FROM EXAM RESOURCE FILE
1543
DELIVER EXAM
1545

FIG. 31

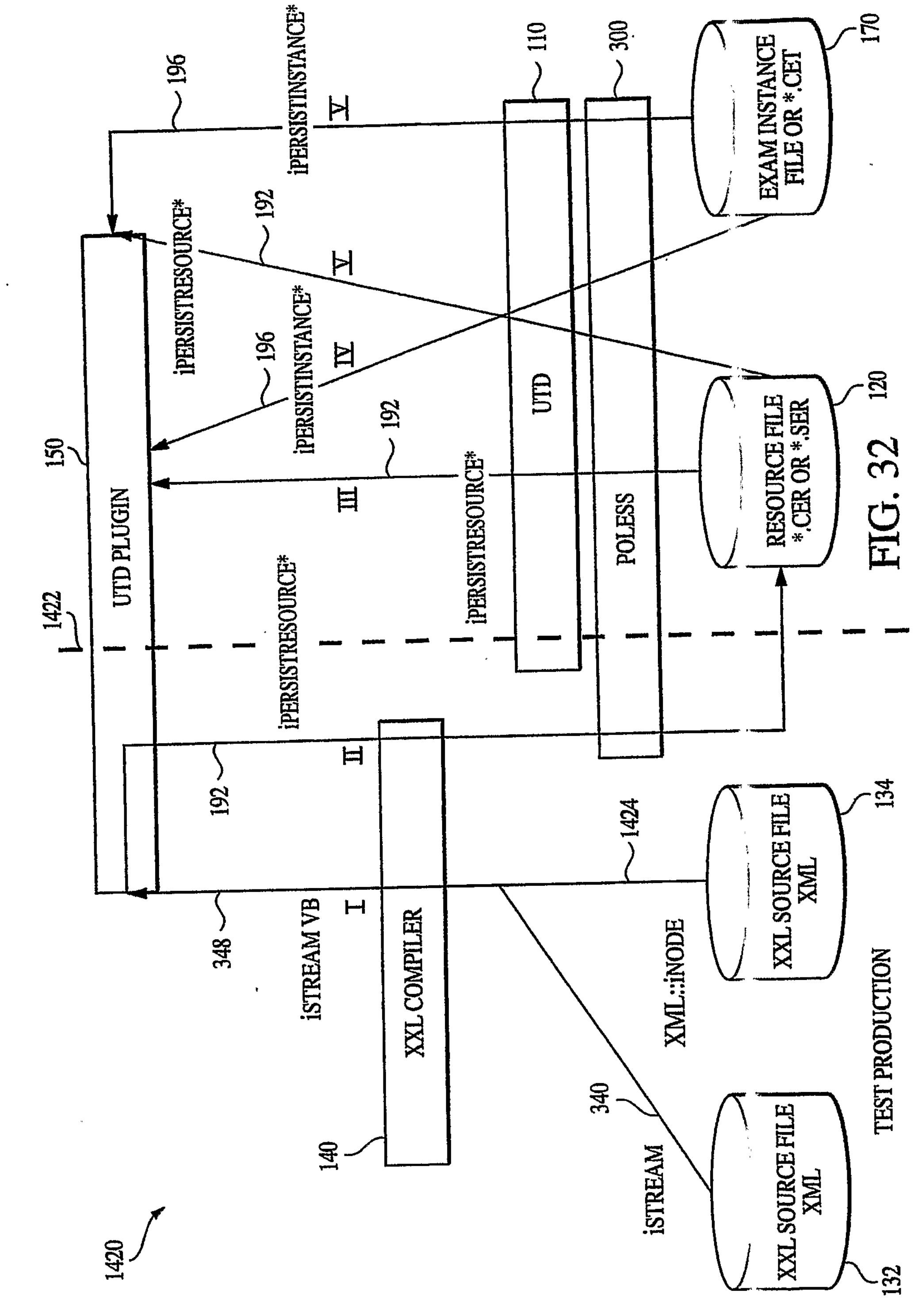

1420
UTD PLUGIN
150
1422
XXL COMPILER
140
UTD
110
POLESS
300
XXL SOURCE FILE XML
132
XXL SOURCE FILE XML
134
RESOURCE FILE *.CER OR *.SER
120
EXAM INSTANCE FILE OR *.CET
170
iSTREAM
340
XML::iNODE
1424
iSTREAM VB
348
I
II
192
iPERSISTRESOURCE*
III
192
iPERSISTRESOURCE*
IV
196
iPERSISTINSTANCE*
V
192
iPERSISTRESOURCE*
V
196
iPERSISTINSTANCE*
TEST PRODUCTION

FIG. 32

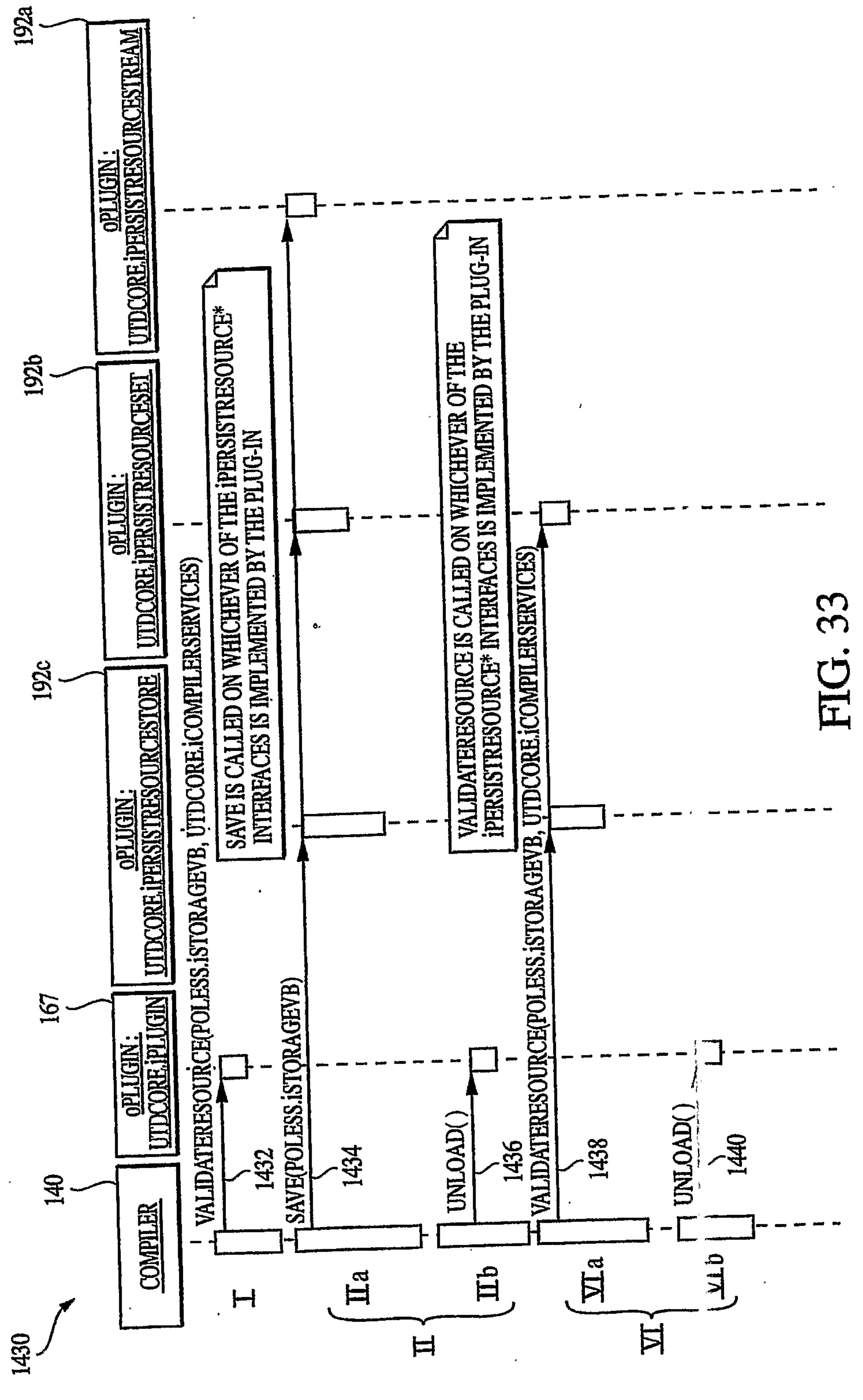

1430
140
COMPILER
167
oPLUGIN : UTDCORE.iPLUGIN
192c
oPLUGIN : UTDCORE.iPERSISTRESOURCESTORE
192b
oPLUGIN : UTDCORE.iPERSISTRESOURCESET
192a
oPLUGIN : UTDCORE.iPERSISTRESOURCESTREAM
VALIDATERESOURCE(POLESS.iSTORAGEVB, UTDCORE.iCOMPILERSERVICES)
1432
SAVE IS CALLED ON WHICHEVER OF THE iPERSISTRESOURCE* INTERFACES IS IMPLEMENTED BY THE PLUG-IN
SAVE(POLESS.iSTORAGEVB)
1434
UNLOAD( )
1436
VALIDATERESOURCE IS CALLED ON WHICHEVER OF THE iPERSISTRESOURCE* INTERFACES IS IMPLEMENTED BY THE PLUG-IN
VALIDATERESOURCE(POLESS.iSTORAGEVB, UTDCORE.iCOMPILERSERVICES)
1438
UNLOAD( )
1440
I
IIa
IIb
II
VIa
VIb
VI

FIG. 33

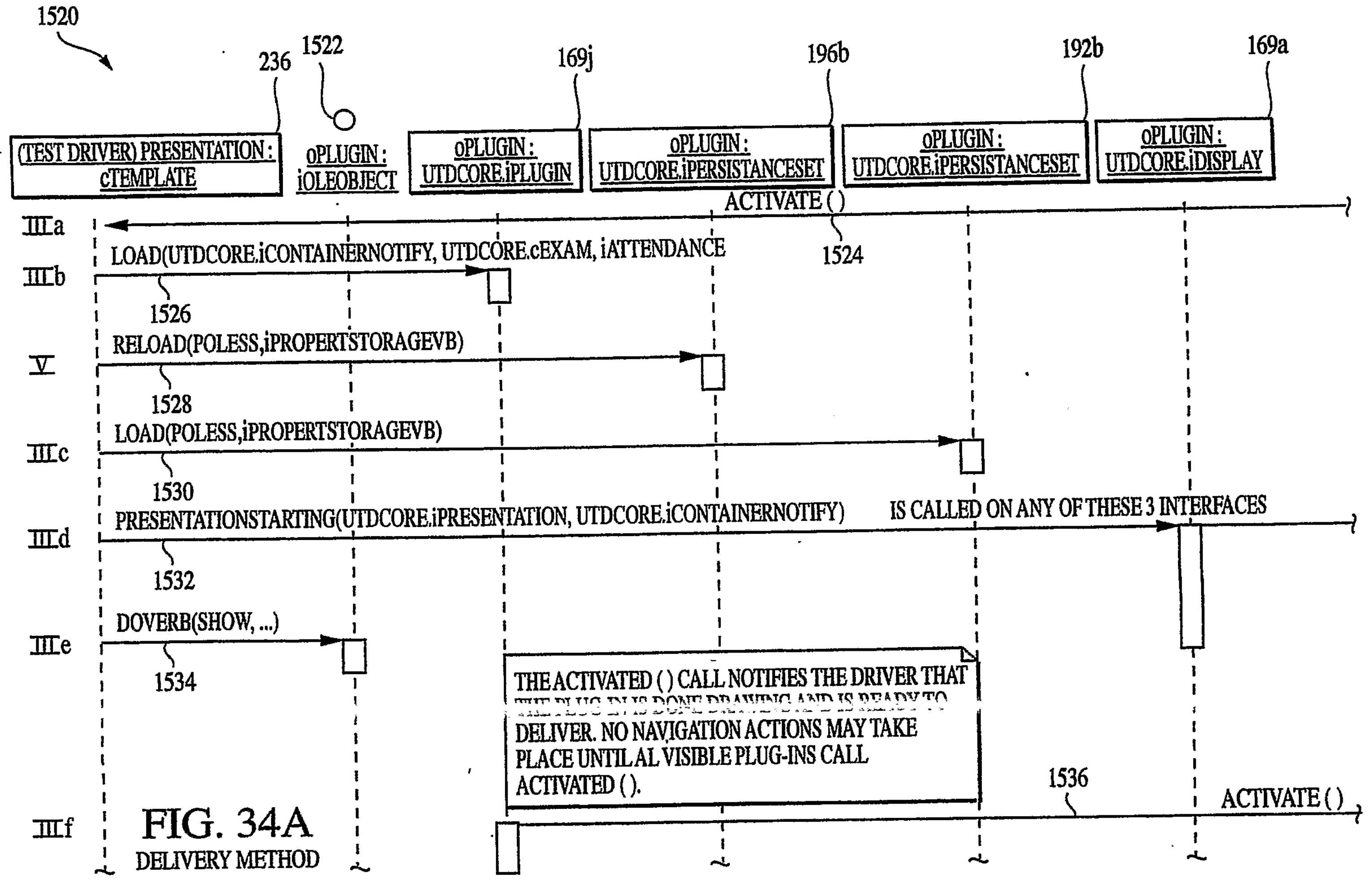

1520
236
1522
169j
196b
192b
169a
(TEST DRIVER) PRESENTATION : cTEMPLATE
oPLUGIN : iOLEOBJECT
oPLUGIN : UTDCORE.iPLUGIN
oPLUGIN : UTDCORE.iPERSISTANCESET
oPLUGIN : UTDCORE.iPERSISTANCESET
oPLUGIN : UTDCORE.iDISPLAY
ACTIVATE ( )
1524
IIIa
IIIb
LOAD(UTDCORE.iCONTAINERNOTIFY, UTDCORE.cEXAM, iATTENDANCE
1526
V
RELOAD(POLESS,iPROPERTSTORAGEVB)
1528
IIIc
LOAD(POLESS,iPROPERTSTORAGEVB)
1530
IIId
PRESENTATIONSTARTING(UTDCORE.iPRESENTATION, UTDCORE.iCONTAINERNOTIFY)
IS CALLED ON ANY OF THESE 3 INTERFACES
1532
IIIe
DOVERB(SHOW, ...)
1534
THE ACTIVATED ( ) CALL NOTIFIES THE DRIVER THAT THE PLUG-IN IS DONE DRAWING AND IS READY TO DELIVER. NO NAVIGATION ACTIONS MAY TAKE PLACE UNTIL AL VISIBLE PLUG-INS CALL ACTIVATED ( ).
1536
ACTIVATE ( )
IIIf

FIG. 34A
DELIVERY METHOD

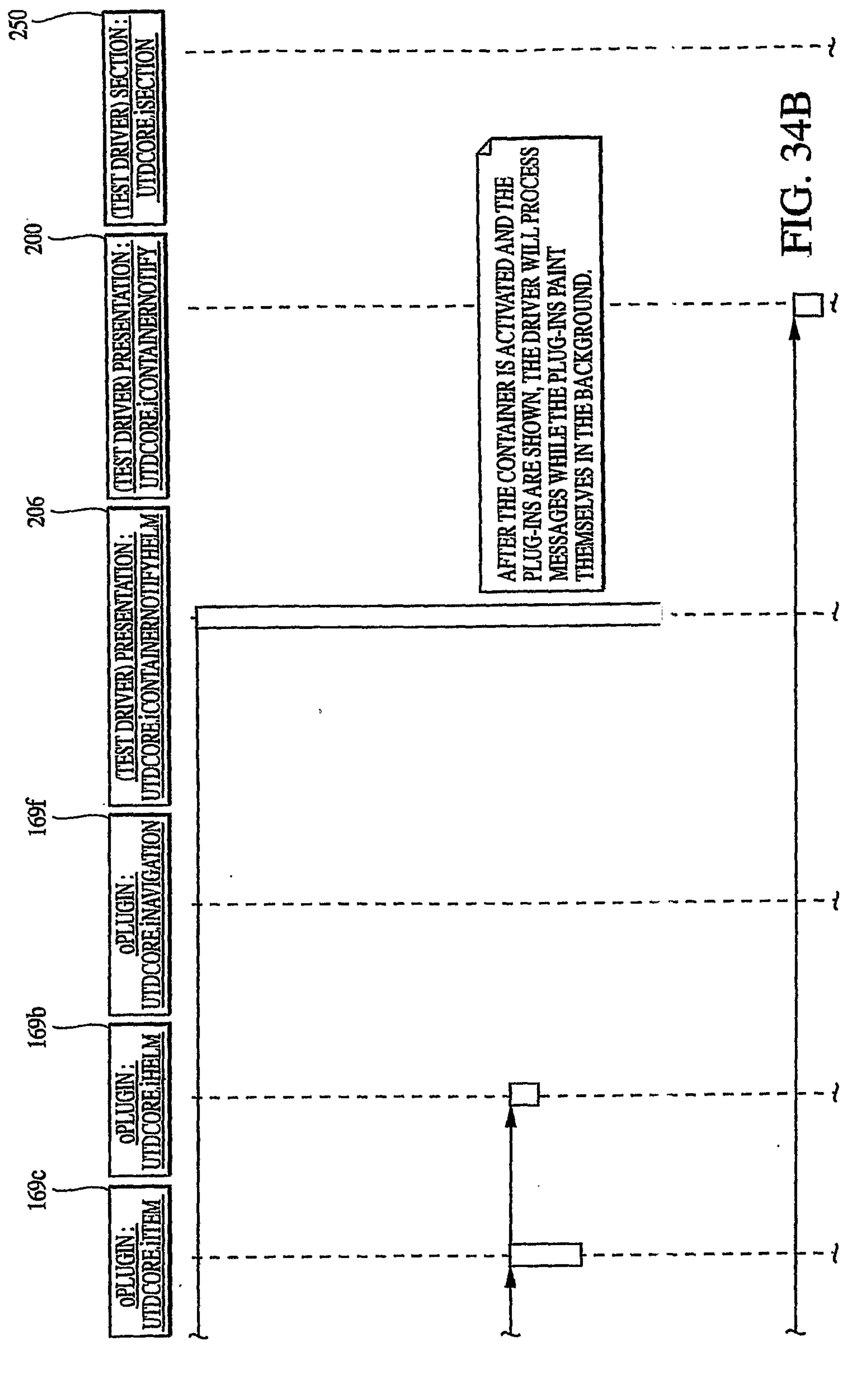

250
(TEST DRIVER) SECTION : UTDCORE.ISECTION
200
(TEST DRIVER) PRESENTATION : UTDCORE.ICONTAINERNOTIFY
206
(TEST DRIVER) PRESENTATION : UTDCORE.ICONTAINERNOTIFYHELM
169f
oPLUGIN : UTDCORE.INAVIGATION
169b
oPLUGIN : UTDCORE.IHELM
169c
oPLUGIN : UTDCORE.IITEM
AFTER THE CONTAINER IS ACTIVATED AND THE PLUG-INS ARE SHOWN, THE DRIVER WILL PROCESS MESSAGES WHILE THE PLUG-INS PAINT THEMSELVES IN THE BACKGROUND.

FIG. 34B

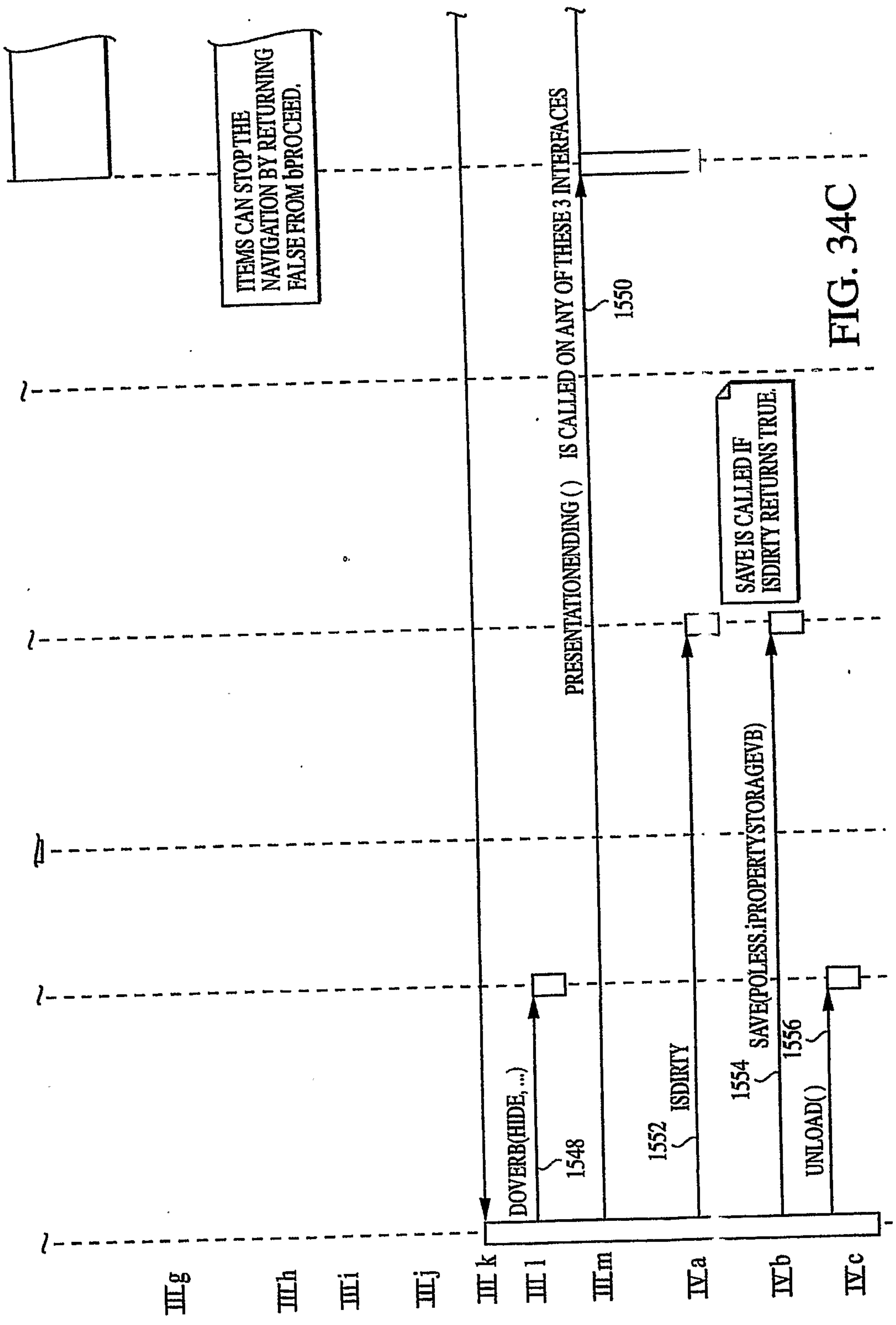
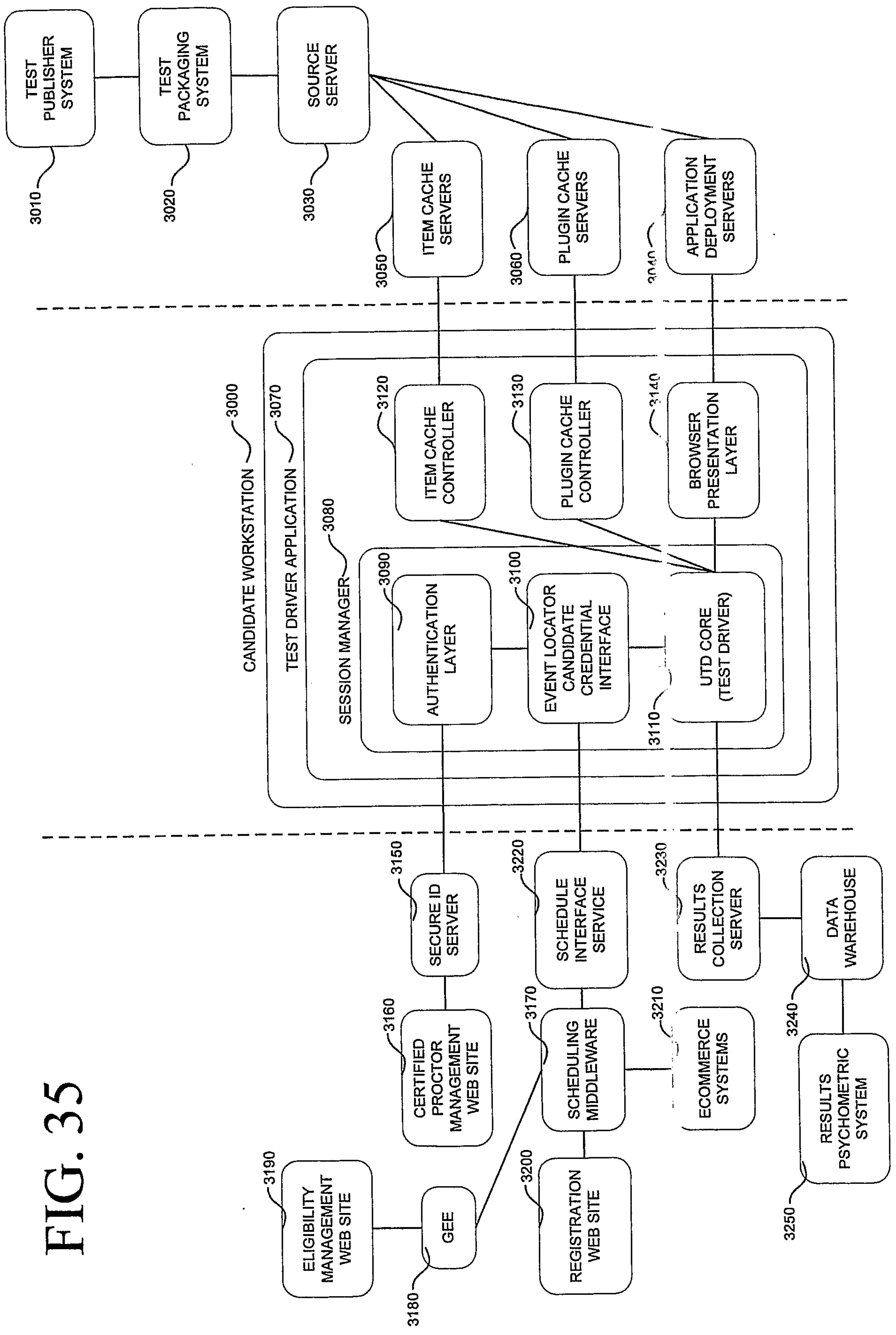

III g
III h
III i
III j
III k
III l
III m
IV a
IV b
IV c
ITEMS CAN STOP THE NAVIGATION BY RETURNING FALSE FROM bPROCEED.
PRESENTATIONENDING ( ) IS CALLED ON ANY OF THESE 3 INTERFACES
1550
SAVE IS CALLED IF ISDIRTY RETURNS TRUE.
DOVERB(HIDE, ...)
1548
ISDIRTY
1552
SAVE(POLESS.iPROPERTYSTORAGEVB)
1554
UNLOAD( )
1556

FIG. 34C

FIG. 35



TEST PUBLISHER SYSTEM
3010
TEST PACKAGING SYSTEM
3020
SOURCE SERVER
3030
ITEM CACHE SERVERS
3050
PLUGIN CACHE SERVERS
3060
APPLICATION DEPLOYMENT SERVERS
3040
CANDIDATE WORKSTATION
3000
TEST DRIVER APPLICATION
3070
SESSION MANAGER
3080
ITEM CACHE CONTROLLER
3120
PLUGIN CACHE CONTROLLER
3130
BROWSER PRESENTATION LAYER
3140
AUTHENTICATION LAYER
3090
EVENT LOCATOR CANDIDATE CREDENTIAL INTERFACE
3100
UTD CORE (TEST DRIVER)
3110
SECURE ID SERVER
3150
SCHEDULE INTERFACE SERVICE
3220
RESULTS COLLECTION SERVER
3230
DATA WAREHOUSE
3240
CERTIFIED PROCTOR MANAGEMENT WEB SITE
3160
SCHEDULING MIDDLEWARE
3170
ECOMMERCE SYSTEMS
3210
RESULTS PSYCHOMETRIC SYSTEM
3250
ELIGIBILITY MANAGEMENT WEB SITE
3190
GEE
3180
REGISTRATION WEB SITE
3200

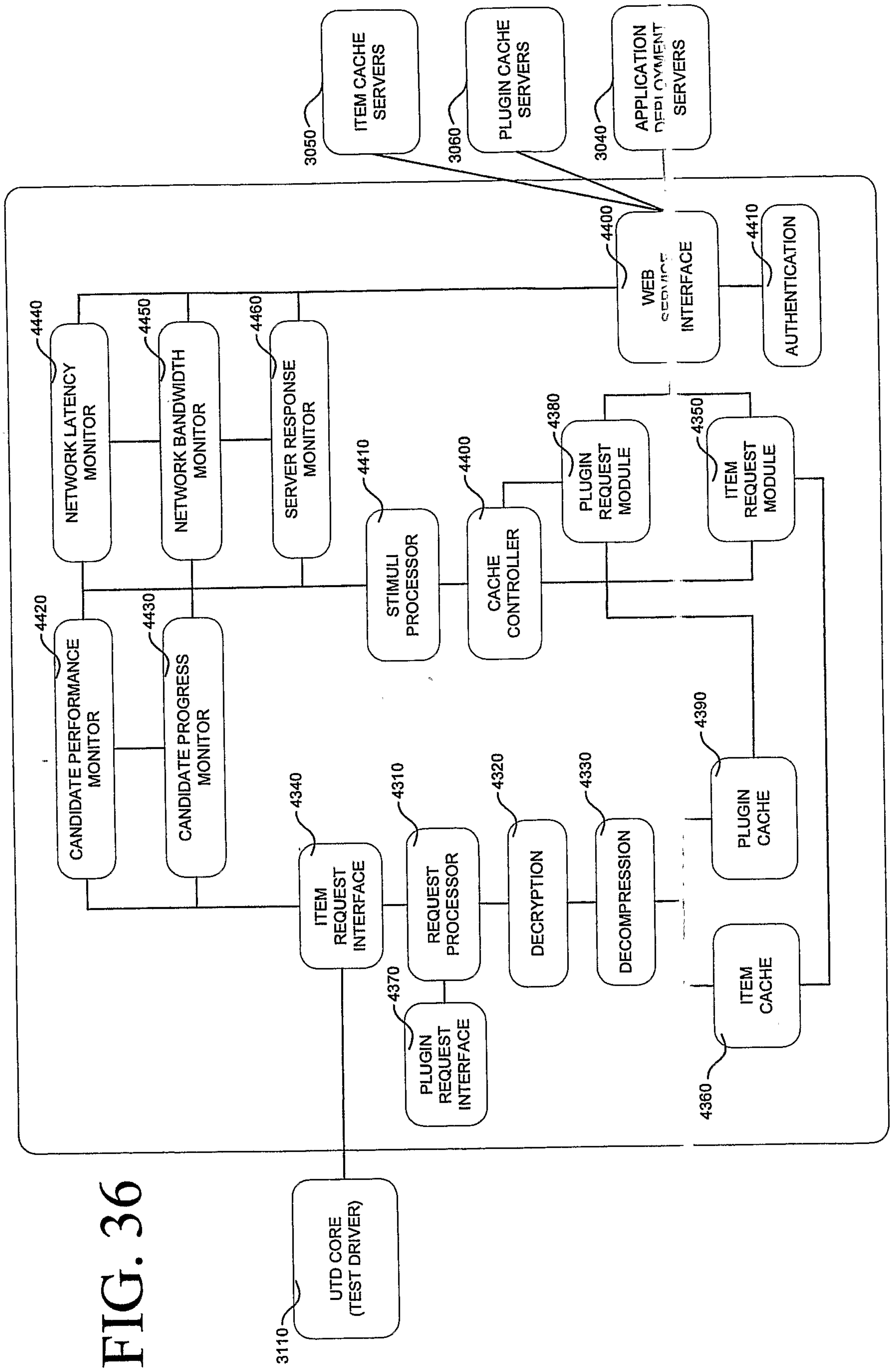

ITEM CACHE SERVERS
3050
PLUGIN CACHE SERVERS
3060
APPLICATION DEPLOYMENT SERVERS
3040
WEB SERVICE INTERFACE
4400
AUTHENTICATION
4410
NETWORK LATENCY MONITOR
4440
NETWORK BANDWIDTH MONITOR
4450
SERVER RESPONSE MONITOR
4460
STIMULI PROCESSOR
4410
CACHE CONTROLLER
4400
PLUGIN REQUEST MODULE
4380
ITEM REQUEST MODULE
4350
CANDIDATE PERFORMANCE MONITOR
4420
CANDIDATE PROGRESS MONITOR
4430
ITEM REQUEST INTERFACE
4340
REQUEST PROCESSOR
4310
DECRYPTION
4320
DECOMPRESSION
4330
PLUGIN CACHE
4390
ITEM CACHE
4360
PLUGIN REQUEST INTERFACE
4370
UTD CORE (TEST DRIVER)
3110

FIG. 36

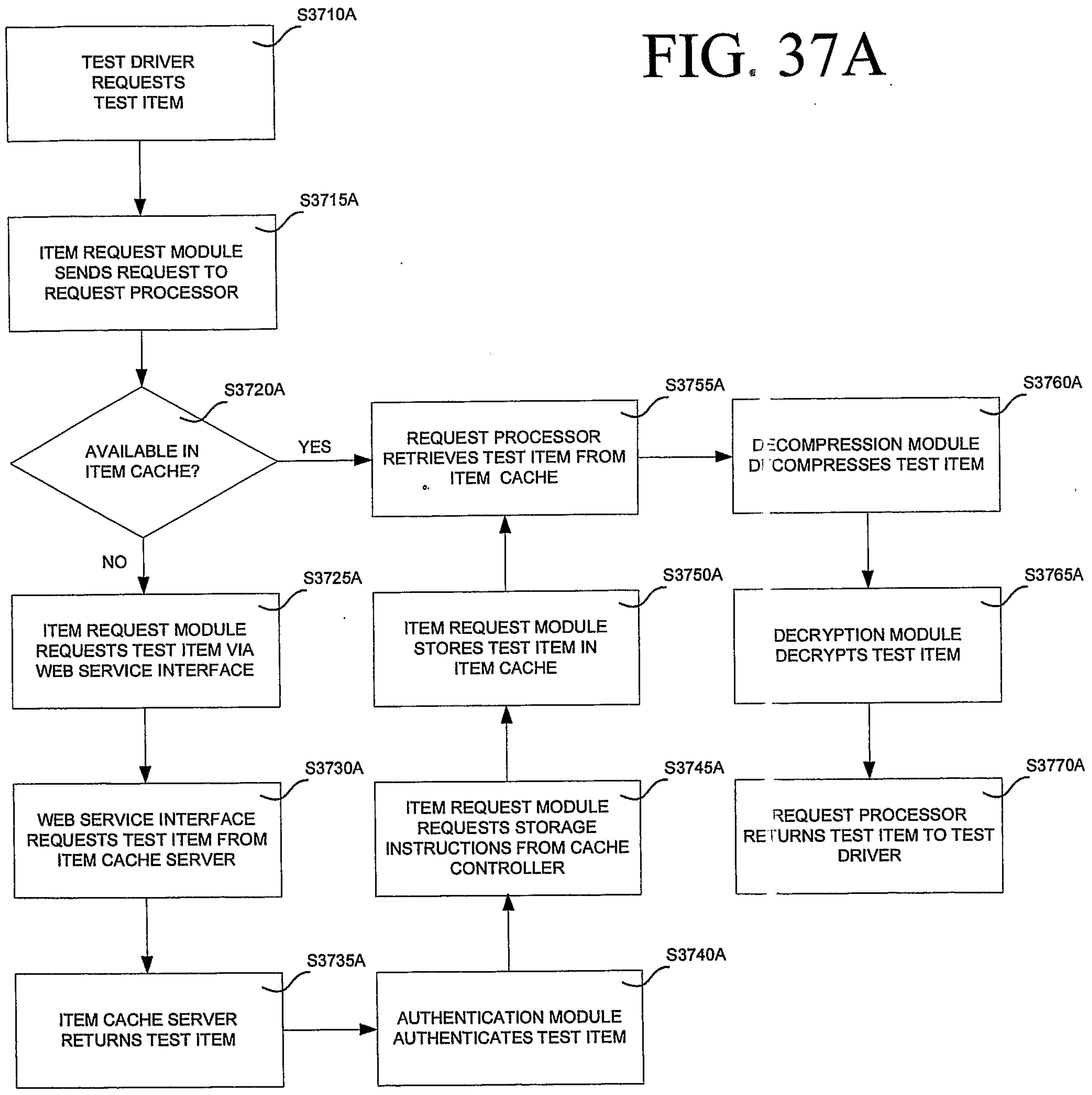

S3710A
TEST DRIVER REQUESTS TEST ITEM
S3715A
ITEM REQUEST MODULE SENDS REQUEST TO REQUEST PROCESSOR
S3720A
AVAILABLE IN ITEM CACHE?
YES
NO
S3725A
ITEM REQUEST MODULE REQUESTS TEST ITEM VIA WEB SERVICE INTERFACE
S3730A
WEB SERVICE INTERFACE REQUESTS TEST ITEM FROM ITEM CACHE SERVER
S3735A
ITEM CACHE SERVER RETURNS TEST ITEM
S3740A
AUTHENTICATION MODULE AUTHENTICATES TEST ITEM
S3745A
ITEM REQUEST MODULE REQUESTS STORAGE INSTRUCTIONS FROM CACHE CONTROLLER
S3750A
ITEM REQUEST MODULE STORES TEST ITEM IN ITEM CACHE
S3755A
REQUEST PROCESSOR RETRIEVES TEST ITEM FROM ITEM CACHE
S3760A
DECOMPRESSION MODULE DECOMPRESSES TEST ITEM
S3765A
DECRYPTION MODULE DECRYPTS TEST ITEM
S3770A
REQUEST PROCESSOR RETURNS TEST ITEM TO TEST DRIVER

FIG. 37A

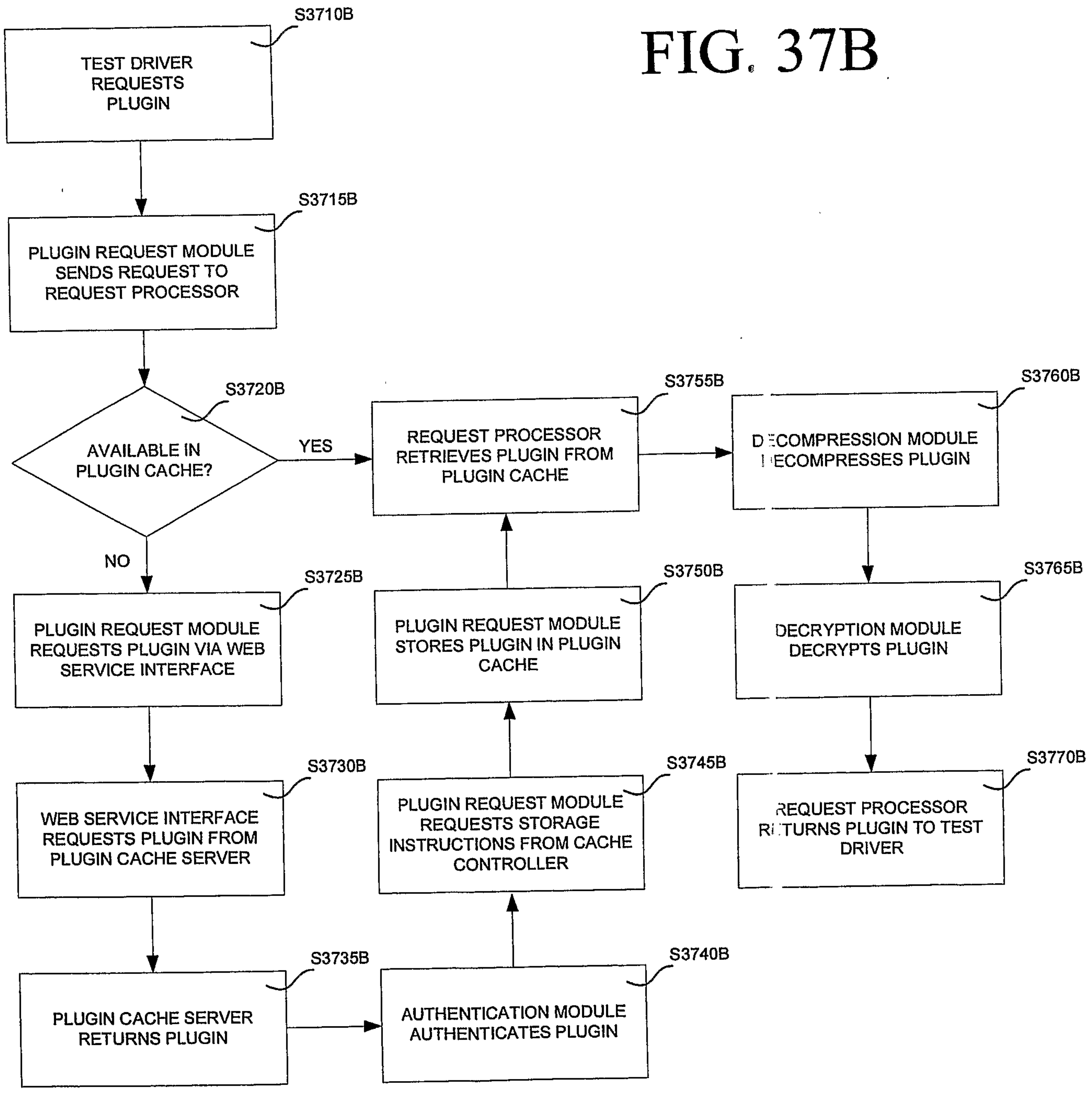

S3710B
TEST DRIVER REQUESTS PLUGIN
S3715B
PLUGIN REQUEST MODULE SENDS REQUEST TO REQUEST PROCESSOR
S3720B
AVAILABLE IN PLUGIN CACHE?
YES
NO
S3725B
PLUGIN REQUEST MODULE REQUESTS PLUGIN VIA WEB SERVICE INTERFACE
S3730B
WEB SERVICE INTERFACE REQUESTS PLUGIN FROM PLUGIN CACHE SERVER
S3735B
PLUGIN CACHE SERVER RETURNS PLUGIN
S3740B
AUTHENTICATION MODULE AUTHENTICATES PLUGIN
S3745B
PLUGIN REQUEST MODULE REQUESTS STORAGE INSTRUCTIONS FROM CACHE CONTROLLER
S3750B
PLUGIN REQUEST MODULE STORES PLUGIN IN PLUGIN CACHE
S3755B
REQUEST PROCESSOR RETRIEVES PLUGIN FROM PLUGIN CACHE
S3760B
DECOMPRESSION MODULE DECOMPRESSES PLUGIN
S3765B
DECRYPTION MODULE DECRYPTS PLUGIN
S3770B
REQUEST PROCESSOR RETURNS PLUGIN TO TEST DRIVER

FIG. 37B

FIG. 38A

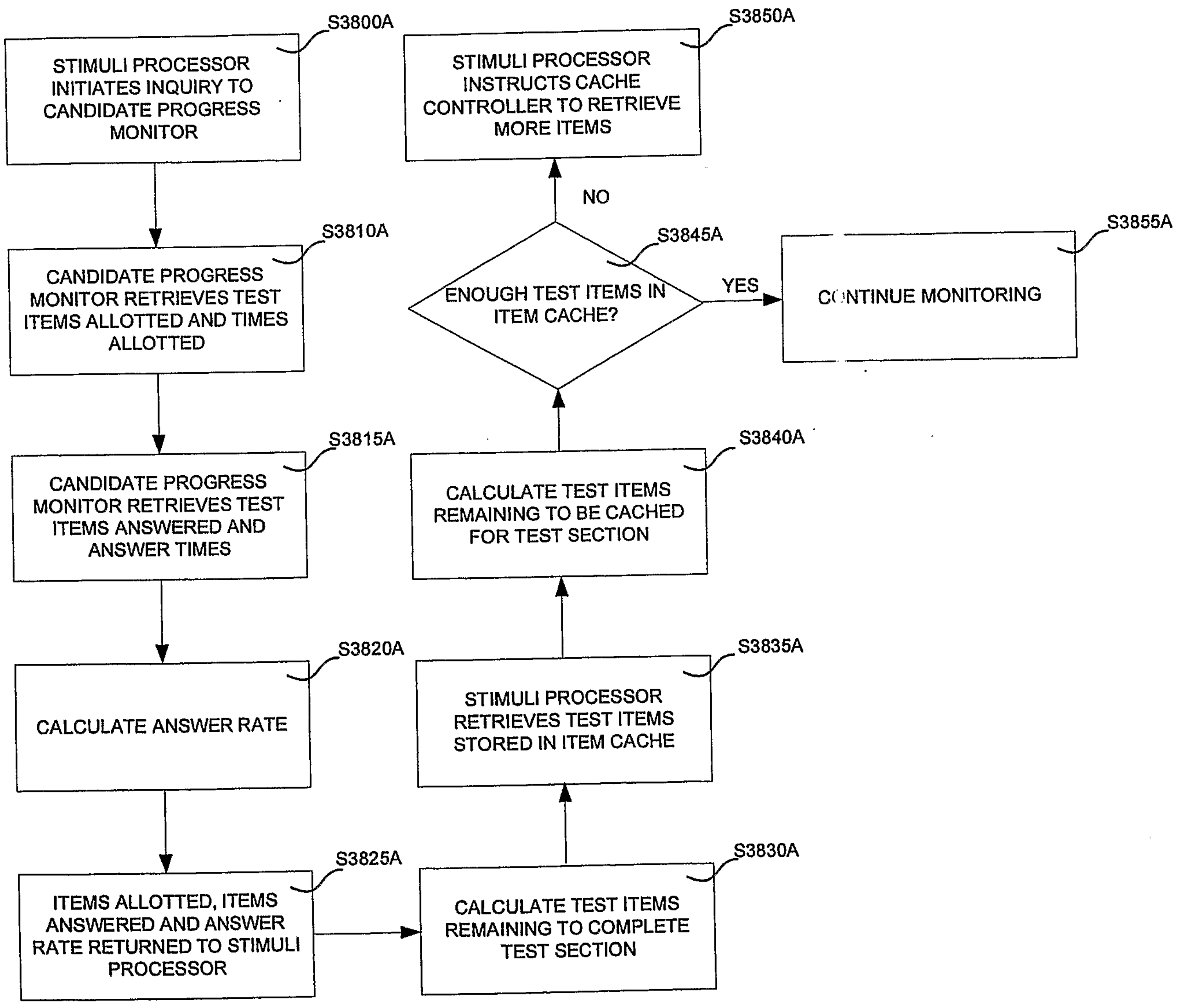

S3800A
STIMULI PROCESSOR INITIATES INQUIRY TO CANDIDATE PROGRESS MONITOR
S3810A
CANDIDATE PROGRESS MONITOR RETRIEVES TEST ITEMS ALLOTTED AND TIMES ALLOTTED
S3815A
CANDIDATE PROGRESS MONITOR RETRIEVES TEST ITEMS ANSWERED AND ANSWER TIMES
S3820A
CALCULATE ANSWER RATE
S3825A
ITEMS ALLOTTED, ITEMS ANSWERED AND ANSWER RATE RETURNED TO STIMULI PROCESSOR
S3830A
CALCULATE TEST ITEMS REMAINING TO COMPLETE TEST SECTION
S3835A
STIMULI PROCESSOR RETRIEVES TEST ITEMS STORED IN ITEM CACHE
S3840A
CALCULATE TEST ITEMS REMAINING TO BE CACHED FOR TEST SECTION
S3845A
ENOUGH TEST ITEMS IN ITEM CACHE?
NO
YES
S3850A
STIMULI PROCESSOR INSTRUCTS CACHE CONTROLLER TO RETRIEVE MORE ITEMS
S3855A
CONTINUE MONITORING

FIG. 38B

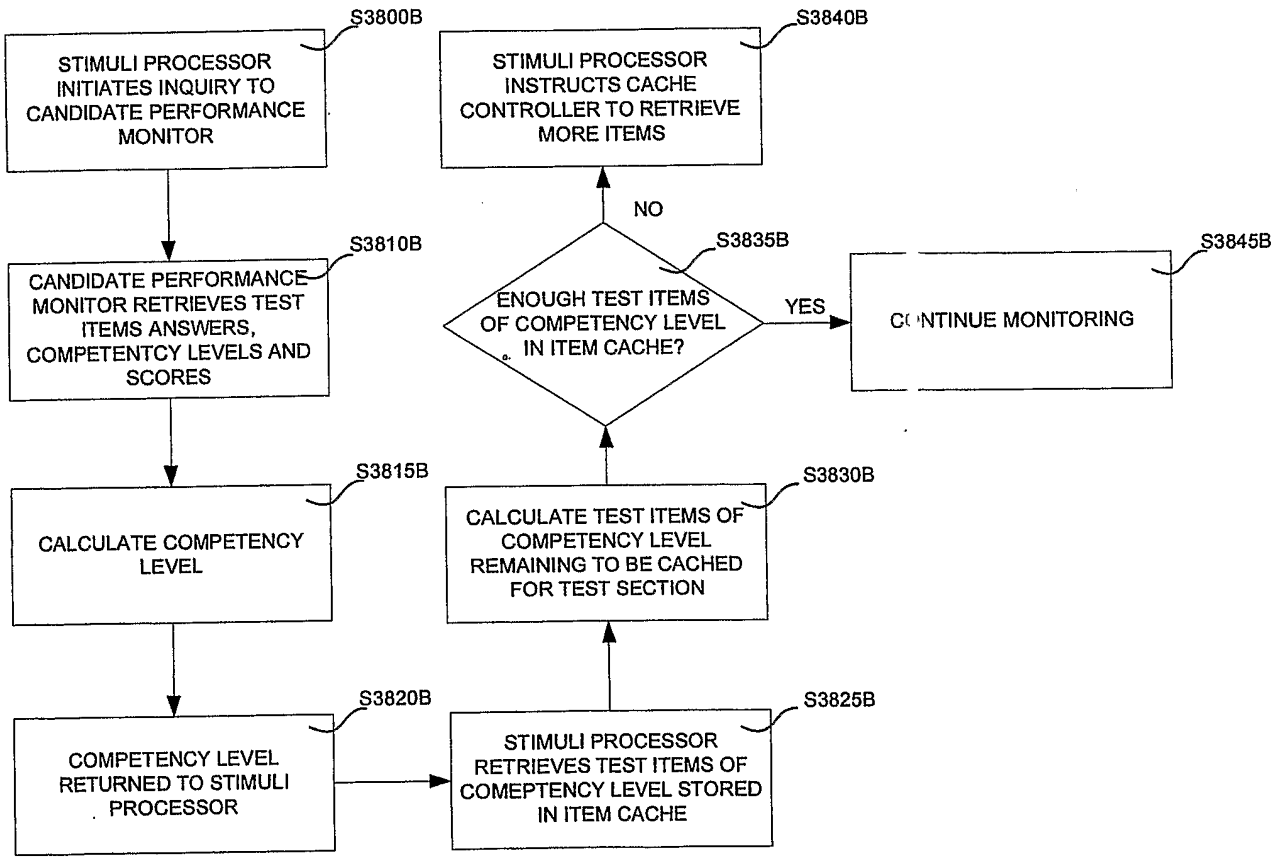

S3800B
STIMULI PROCESSOR INITIATES INQUIRY TO CANDIDATE PERFORMANCE MONITOR
S3810B
CANDIDATE PERFORMANCE MONITOR RETRIEVES TEST ITEMS ANSWERS, COMPETENTCY LEVELS AND SCORES
S3815B
CALCULATE COMPETENCY LEVEL
S3820B
COMPETENCY LEVEL RETURNED TO STIMULI PROCESSOR
S3825B
STIMULI PROCESSOR RETRIEVES TEST ITEMS OF COMEPTENCY LEVEL STORED IN ITEM CACHE
S3830B
CALCULATE TEST ITEMS OF COMPETENCY LEVEL REMAINING TO BE CACHED FOR TEST SECTION
S3835B
ENOUGH TEST ITEMS OF COMPETENCY LEVEL IN ITEM CACHE?
NO
S3840B
STIMULI PROCESSOR INSTRUCTS CACHE CONTROLLER TO RETRIEVE MORE ITEMS
YES
S3845B
CONTINUE MONITORING

FIG. 38C

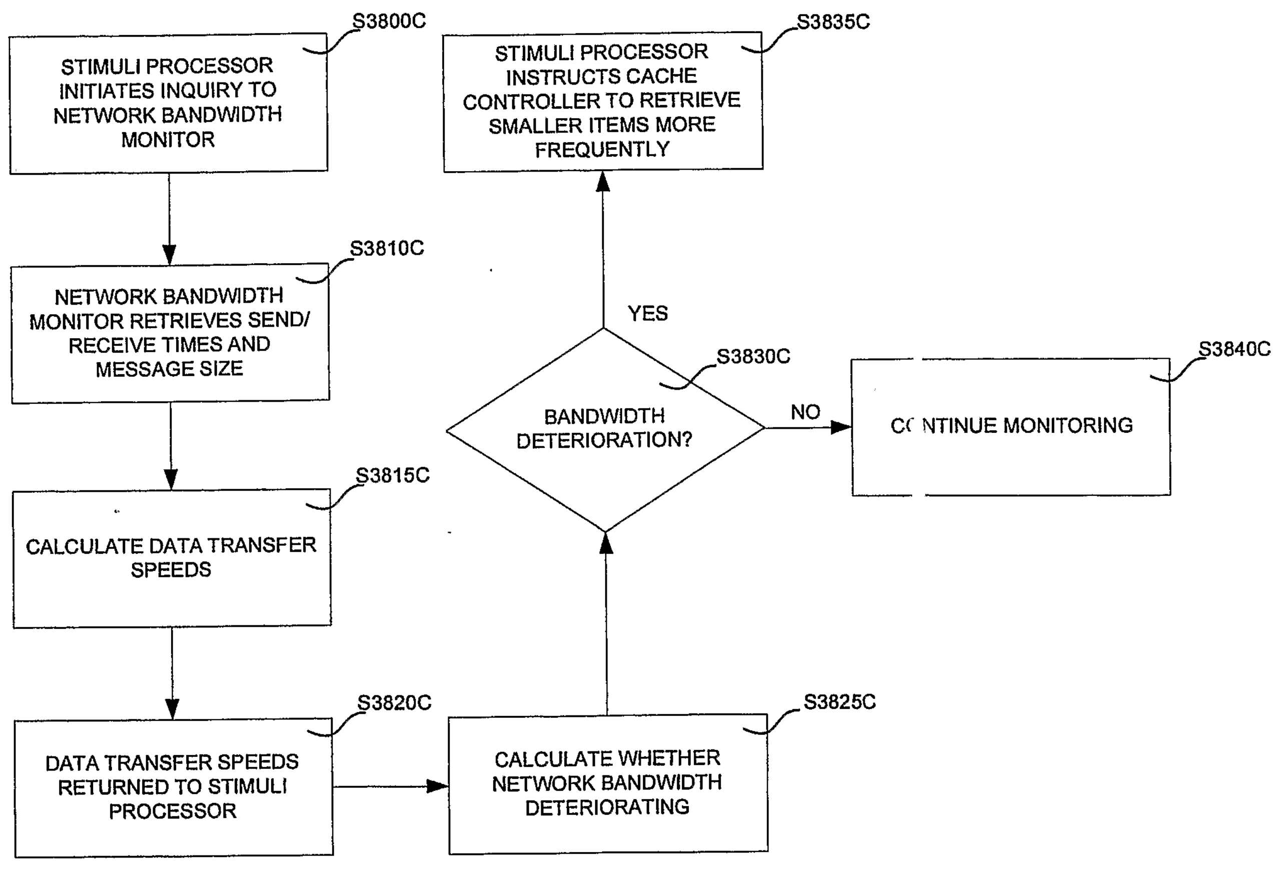

STIMULI PROCESSOR INITIATES INQUIRY TO NETWORK BANDWIDTH MONITOR
S3800C
NETWORK BANDWIDTH MONITOR RETRIEVES SEND/ RECEIVE TIMES AND MESSAGE SIZE
S3810C
CALCULATE DATA TRANSFER SPEEDS
S3815C
DATA TRANSFER SPEEDS RETURNED TO STIMULI PROCESSOR
S3820C
CALCULATE WHETHER NETWORK BANDWIDTH DETERIORATING
S3825C
BANDWIDTH DETERIORATION?
S3830C
YES
NO
STIMULI PROCESSOR INSTRUCTS CACHE CONTROLLER TO RETRIEVE SMALLER ITEMS MORE FREQUENTLY
S3835C
CONTINUE MONITORING
S3840C

FIG. 38D

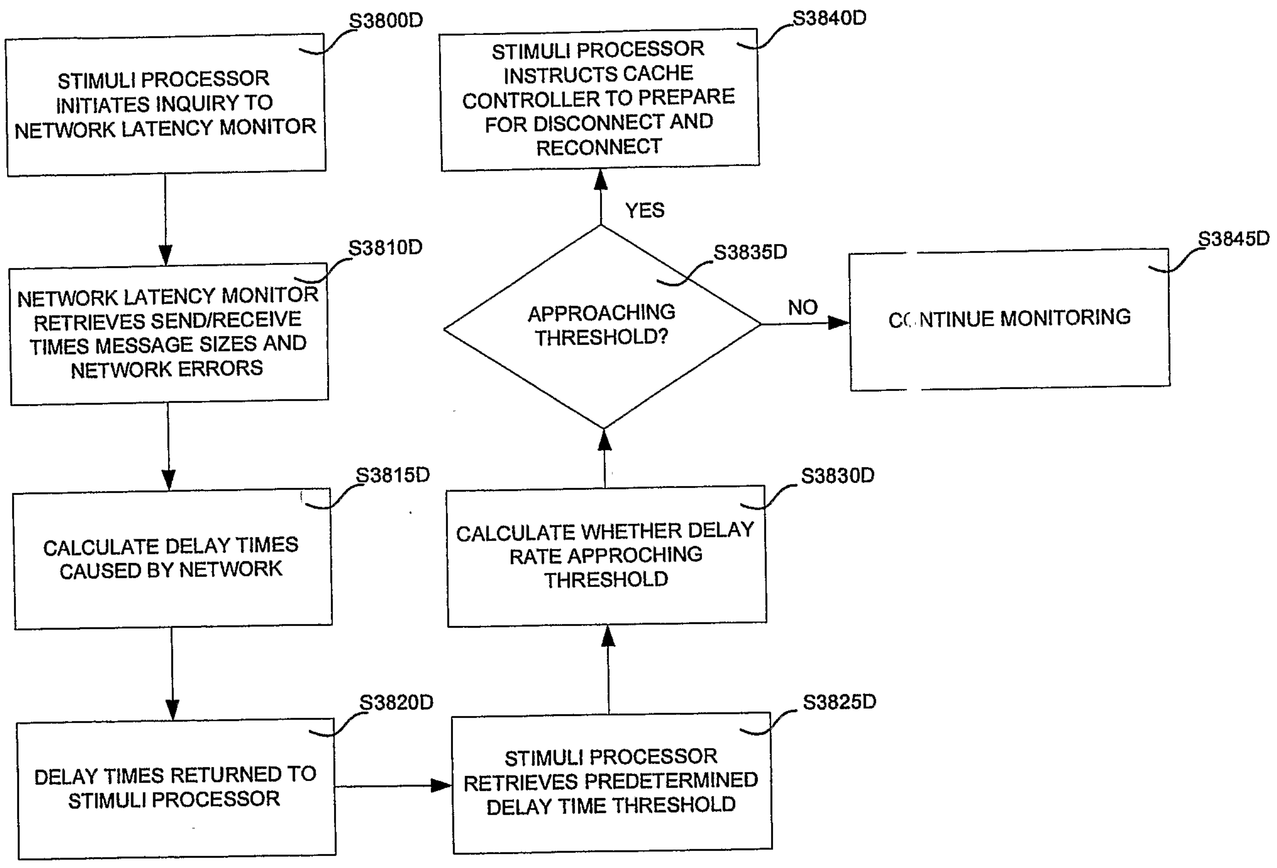
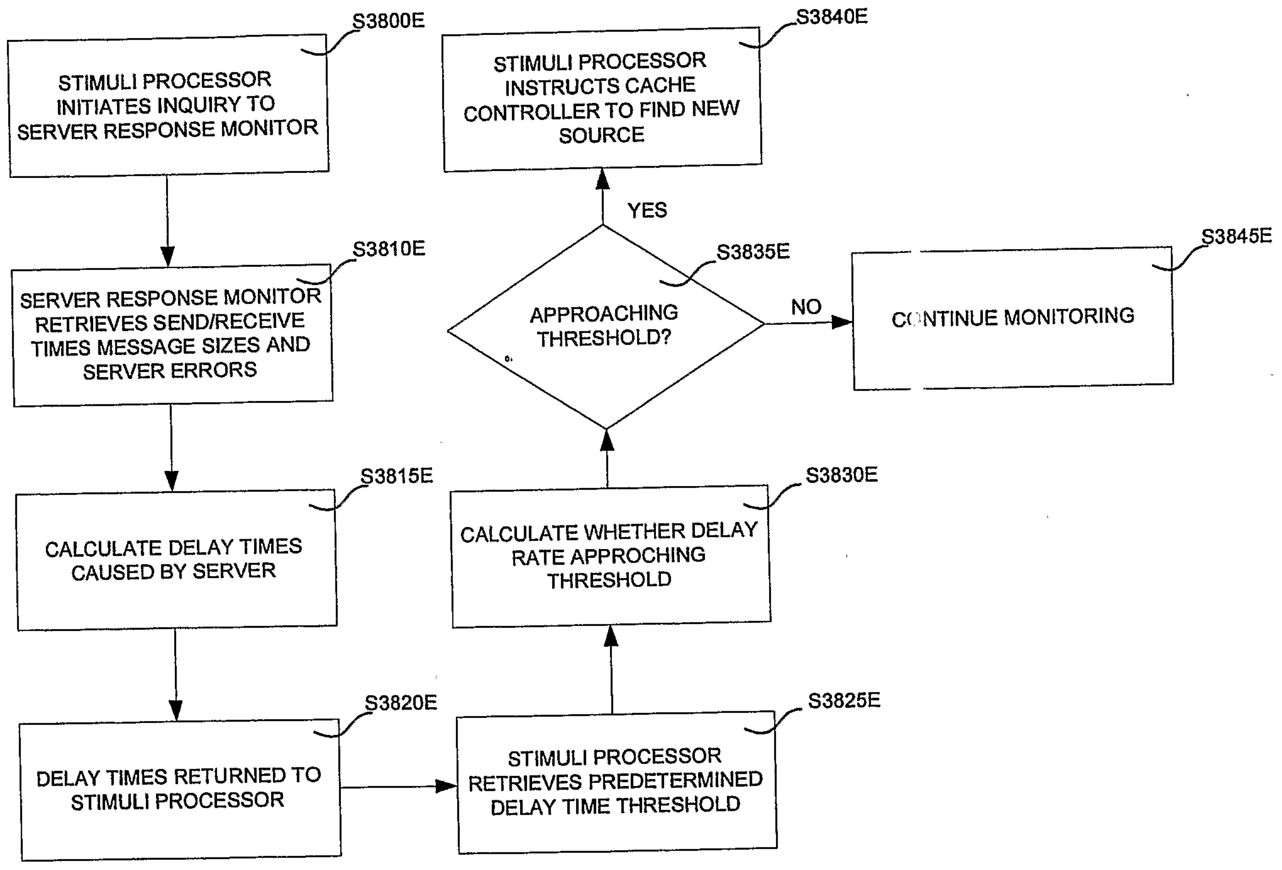

S3800D
STIMULI PROCESSOR INITIATES INQUIRY TO NETWORK LATENCY MONITOR
S3810D
NETWORK LATENCY MONITOR RETRIEVES SEND/RECEIVE TIMES MESSAGE SIZES AND NETWORK ERRORS
S3815D
CALCULATE DELAY TIMES CAUSED BY NETWORK
S3820D
DELAY TIMES RETURNED TO STIMULI PROCESSOR
S3825D
STIMULI PROCESSOR RETRIEVES PREDETERMINED DELAY TIME THRESHOLD
S3830D
CALCULATE WHETHER DELAY RATE APPROCHING THRESHOLD
S3835D
APPROACHING THRESHOLD?
YES
NO
S3840D
STIMULI PROCESSOR INSTRUCTS CACHE CONTROLLER TO PREPARE FOR DISCONNECT AND RECONNECT
S3845D
CONTINUE MONITORING

FIG. 38E


S3800E
STIMULI PROCESSOR INITIATES INQUIRY TO SERVER RESPONSE MONITOR
S3810E
SERVER RESPONSE MONITOR RETRIEVES SEND/RECEIVE TIMES MESSAGE SIZES AND SERVER ERRORS
S3815E
CALCULATE DELAY TIMES CAUSED BY SERVER
S3820E
DELAY TIMES RETURNED TO STIMULI PROCESSOR
S3825E
STIMULI PROCESSOR RETRIEVES PREDETERMINED DELAY TIME THRESHOLD
S3830E
CALCULATE WHETHER DELAY RATE APPROCHING THRESHOLD
S3835E
APPROACHING THRESHOLD?
YES
S3840E
STIMULI PROCESSOR INSTRUCTS CACHE CONTROLLER TO FIND NEW SOURCE
NO
S3845E
CONTINUE MONITORING