[Continued on next page]

Published:
— with international search report
— before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments

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.
Website Checking Tool

The present invention relates to a website checking tool and more particularly to such a tool for checking the validity and accessibility of a website.

The number of websites provided by service providers and others both via the so-called world-wide-web (WWW) or internet and on company intranets has expanded rapidly. Many such websites contain many hundreds or thousands of pages of information which may include text, graphics, links to other pages on the same or other websites, applets, java scripts and so on. Checking the content of every page of a website may not be practical in most cases and it is only when a user reports faults such as a broken link to another page that errors may be discovered. Since most users will not return comments on broken links but will simply surf elsewhere it is possible that potential custom is lost to the website owner or that potentially useful information to the user is missed.

More recently it has been realised that people with different abilities also find internet or intranet access extremely useful but the presence of certain features can adversely affect their enjoyment of the experience. For example, the presence of a photograph (for example a .jpg image), a video (.avi for example) or an applet may not be compatible with an audible text reading device to assist users with impaired viewing ability. In addition to visual impairment users may have hearing impairments, motor impairments or cognitive impairments.

Recent legislation in many countries aims to facilitate the use of the internet by differently abled people for example by providing alternative text descriptors with a photograph. Again, however, checking a website for compliance with regulations or to ensure that the website is capable of serving the needs of the differently abled may be too complex for a human checker to do.

An accessible web site contains information that is available to everyone, including people with disabilities. Hence an accessible web site must not be dependent upon the rendering of design elements such as colour, font size or layout (which can be overridden in the browser settings) and it must be compatible with assistive technologies that may be used by people with disabilities.

Making web-sites accessible is often misunderstood to just refer to aiding people who have visual impairments, i.e. those who may have colour turned off and use large font sizes or may be using text to speech browsers. However, any disability should be considered, including:

- Visual impairments
• Hearing impairments
• Motor impairments
• Cognitive impairments

Each of the above includes a wide range of conditions which can be permanent or temporary and can range from e.g. partial sight and colour blindness through total blindness/deafness to conditions such as epilepsy and cerebral palsy.

In 1994 the World Wide Web Consortium (W3C) began an investigation into the accessibility of web-sites to people with disabilities and so the Web Accessibility Initiative (WAI) was formed. The WAI have produced several documents, the most widely referred to being the Web Content Accessibility Guidelines (WCAG) - 14 separate guidelines that contain checkpoints designated as priority 1, priority 2 and priority 3:
• Priority 1: actions that website designers must take for a site to be accessible.
• Priority 2: actions that website designers should take for a site to be accessible.
• Priority 3: actions that website designers might take to improve website accessibility.

The W3C are the source of accessibility information and guidelines that form the basis of the legislation in most countries. The W3C have a comprehensive, though sometimes confusing/contradictory, set of documentation on website accessibility.

W3C documentation which is constantly updated in line with current thinking can be accessed through the world-wide web at http://www.w3.org/. At the date of preparation of this specification, the WCAG was accessible at http://www.w3.org/TR/WCAG10/full-checklist.html.

According to the present invention there is provided a website checking tool including processing means responsive to input of a website address to cause download of information from the page specified by the website address, the processing means:

Storing the website address,
scanning the downloaded information,
determining the content and format of the downloaded information and classifying elements found therein according to type,

storing a respective list for a plurality of types of downloaded information and data associated therewith and,

for each of the plurality of types of information testing each item in the respective list for compatibility with predetermined characteristics and/or operations, by determining from responses at each stage the compatibility of the information with the predetermined characteristics and/or operations and storing data
identifying incompatibilities and associating said stored data with the input website address,

at least one of said types being characterised by a being a URL or address of linked information, each such URL or address being stored in the corresponding list for its type and being checked for response when accessed, whereby incompatible information on a website identified by an input website address is identified.

A website checking tool in accordance with the invention will now be described with reference to the accompanying drawings of which:

Figures 1 and 2 form a flow chart showing the main processing control steps of the tool engine;

Figure 3 is a flow chart showing the steps of a local file processing macro function of figures 1 & 2;

Figure 4 is a flow chart showing the steps of a remote file processing macro function of figures 1 & 2;

Figure 5 is a flow chart showing the steps of each of a number of threads spawned by the remote file processing macro of figure 4;

Figure 6 is a flow chart showing the steps taken to retrieve files;

Figure 7 is a flow chart showing the steps used to check HTML content of a local page;

Figure 8 is a flow chart of a parsing function of the flow chart of figure 7;

Figure 9 is a state transition diagram of a scheduler service for the tool;

Figure 10 is an entity relationship diagram showing database components used by the scheduler;

Figure 11 shows a process flow chart for the scheduler service;

Figures 12 to 20 form a series of flow charts indicating the steps of various macro functions called by the scheduler service of figure 11;

Figure 21 shows in greater detail one of the functions within the macro function of figures 18 to 20; and

Figure 22 is a block schematic diagram of a suitable apparatus for operating the website checking tool of the invention.

Referring first to figure 22, the tool of the present invention comprises four main components which run on one or more Microsoft Windows (trademark) based file servers 1,2. The primary components are a database 3, scheduler 4, user interface 5 and a command line program engine 6.
The database 3 is an SQL Server database containing configuration standing data and stores all generated results and statistics. It is common to all parts of the system, and provides communications between each component present. While the invention is described with reference to an SQL Server database it will be appreciated that other types of database could be used (for example an Access Database) although functionality of other database types may vary such that support for more complex queries may not be available.

The engine 6 is a Windows command line program, written in C++ and accepts various option parameters, normally read from the database 3, which, at run-time, specifies the integrity checking to be applied to a nominated web-site (by supplied URL). The engine 6 uses hypertext transfer protocol (HTTP) to access each site and page/object and then analyses the returned hypertext markup Language (HTML) comprising each site/page as detailed hereinafter with reference to the flow charts and other drawings.

The user interface 5 provides interactive access to the checking and reporting facilities using IIS Active Server Pages (ASP). It also provides access to a detailed help guide, results guide and pages of answers to Frequently Asked Questions (FAQ).

The Scheduler 4 initiates, controls and monitors all scheduled tasks as well as all site checks that have been entered in its database, either manually by users or by loading the Publisher’s Database (PD) supplied weekly check Comma Separated Value (CSV) format data file. Other database management facilities may be used to check automatically with associated publisher information (for example Microsoft Excel, text editor) another database system containing the data. The system may load a CSV file of the specified format into its database to control periodic automatic site checks.

The system may be programmed to provide both batch (automated) and user interactive (manual) facilities for checking specified web sites. The batch check is entirely driven by site URL and associated publisher email address data provided by the PD for all registered Intranet web-sites.

Each check run over a web-site generates overall run statistics and detailed results for each page analysed in that site. This collected data is stored in an integrated relational database (account database) 9, and can be viewed in a provided set of management and user reports which may be driven by user entered query parameters.

Web sites 7 which are to be checked are accessed by way of an Intranet or the Internet 8 and many will generate a large amount of output, depending upon the complexity of each site and the options chosen. Accordingly at the completion of each run
on a specified website 8 by URL the results reports can be notified by e-mail to the user after either a batch check (always emailed to registered publisher) or a manual check (optional email to user). Alternatively, run progress can be reviewed in real time.

Email reports are actually configurable to control when and to whom the reports generated by the engine and scheduler are transmitted. The above relates to a potential default setting in the absence of request for specific reports. For example the system may be set to email a publisher or franchise holder only if errors or warnings are generated. Franchise holders may request an e-mail report having a check summary of all websites in their controlled area after each batch run.

Turning now additionally to Figures 1 and 3, when the scheduler 4 determines that a website is due for checking (based on entries in the database 3) it will call for an instance of the engine 6 to run to carry out the necessary checks. The instance name is used to look up database connection details from the registry and the schedule check id (see fig 10 table 50) to look at the URL to be checked together with other relevant parameters such as check option and the like. If there is no database information it is possible to run the engine provided all of the parameters are received by the engine 6 in the command line and output is written to the screen for the user or to a printer (not shown in Figure 22) as appropriate. Thus, on start up as indicated at 100 the command line options are received and a check is carried out at 105 to determine whether an instance name (sched_check id Fig 10) is present in the command line options. If the instance name is passed registry details are retrieved from the database 3 including all necessary configuration information to be used by the engine 6.

If the command line options do not include an instance name then the configuration is assumed to be potentially present in the command line options (for example for a manually started check) and once the configuration has been obtained or received then at step 115 the presence of a Database Source Name (DSN) enables run options to be read from the database at 120 together with mime types and domain and proxy names. A Domains_name table whole or part of domain names or IP Addresses of web servers within the network under test effectively defining an intranet under test. Proxy servers are identified in a proxy_servers table so that a set of proxy servers are used cyclically to access sites or pages outside of the defined domains (intranet). If no DSN is present in the system either from the command line option or the configuration downloaded from the registry at step 110, then run options are read from the command line information previously received (step 130). Once the necessary run options have been obtained a check is carried out on the information present to ensure that the engine
6 is ready to run and if it is not as checked at 135, the spawned engine will close down and exit at 140. At step 141 a site entry is created in the database 3 to which the engine 6 will pass details and store information. The site entry data (a URL for example) is used to start the first entry of a directory tree (see fig 23) as a file object in the database at 150 prior to opening a loop (155-165) which checks the information held or accessed by way of the specified URL.

File objects in respect of the site entry in the database include information such as the file size, response codes and whether or not a particular object has been examined. File objects are stored in a tree data structure to minimise memory usage and to ensure that recovery of the required data is as fast as possible. Two trees are created within the data file structure, one for local files and one for remotely accessed files and as files are found by the system they are added to the respective trees if they have not already been examined so that all entries in the two trees are exclusive and unique. The information held in respect of each file reference includes details of where the file reference has been linked from and the position at which it has been linked. But in the case of an existing file already in the work queue, when a further link to that is found an additional reference is added to the information in the tree. Thus, the entry profile reference may be along the lines of, for example, page 1.html, reference from page2.html at tag position 123 (further references to the called page may be added as and when found for example reference from pageN.html at tag position nnn).

Returning to Figure 2, a file is fetched from the work queue 155 and is then processed using the procedure shown in Figure 3 to which reference is now additionally made. Thus, considering the local file processing function, a local file, which has been queued is fetched 300, (on the first occurrence being the URL of the home page of the site host for example) and a check carried out on the work queue at 305 to determine whether the instance is marked as having already been examined. If the instance has not been examined then referring now also to Figure 6 an HTTP fetch 310 is carried out to recover information from the page by way of the Intranet or Internet 8 of Figure 22. Figure 6 is described in further detail hereinafter. Returning then to Figure 3, if the information returned indicates a redirection 315 then the URL of the redirection is added immediately to the file queue for remote or local files as appropriate at 320 and if it is a redirection to a local file the engine handles the response immediately by returning to the start of the loop at 300. If at 325 it is determined that the queued file is stored in the remote file tree and work group the system exits unless there are further local files queued for examination.
Considering briefly the add file process at 320, it is noted that part of the function is to check that the file reference returned is not one that has already been handled. If the returned URL is already present in the respective queue to which the URL refers, any errors returned are flagged and a reference added to the data held in respect of the URL in the respective file queue.

At 330 a brief check is carried out on the returned information if it originally appeared that a non-HTML file was being fetched from the web site that in practice returned an HTML mime type (at 310). If this is the case then the file is re-fetched. Assuming that the HTTP fetch at 310 has returned an HTML page the response code and mime type at 335 is examined and if it is a local file with HTML mime type at 336 then the HTML content is parsed and checked 338 as hereinafter described with reference to Figure 7. Part of the parsing code includes an add file process in respect of each link located and these will be added to the appropriate work queues if the links in question have not already been examined.

A check is now carried out to determine if there are further local files queued and, if so, these are handled in a similar manner by returning from 340 to the fetch queued local file function at 300. The interrogation may be simply of the local file done information that if on recovering the last of the local files in the work group then at 305 the system proceeds to populate local information done 350 and exits from the system to return to step 165 of Figure 2 in which a check ensuring that the local file queue is empty is carried out.

Having carried out all necessary checks on the local files, then checks are carried on remote files as indicated at 170 using the process of Figures 4 and 5 to which reference is now made.

In order to accelerate the process of checking links etc in remote files, a number of threads 410 are started up by the system sharing out the remote files held in the remote files work group across the threads. Thus the URLs present in the remote file work queue are divided up and loaded into individual thread work queues at step 400 and a number of threads are started up at 405. These are indicated as threads 1 to n in Figure 4 at 410. In practice up to twenty threads may be started simultaneously although this should not be considered to be either a maximum or norm. The processing of threads will be described hereinafter with reference to Figure 5 but once all of the threads are completed as indicated at 415, then all of the remote response codes are checked to determine whether, for example, a remote file access returns a file not found indication so that an error is logged against the referring page 416.
Thus referring to Figure 5, each thread 410 starts concurrently with the other threads and creates an HTTP object at 510 and then checks to determine whether any files are left to be worked in the thread 515.

If there are files left to be processed, then at 520 the next file from the remote threads queue is retrieved. If the URL indicates an FTP type URL then an FTP request is sent to the specified remote site at 530 and the response there from is stored. Alternatively, if the indication is not an FTP URL then an HTTP request 535 is made to the remote URL and again the response is stored. Now at 540 it is determined whether the response is a redirection and if not the thread returns to 515 to determine if there are further files to be processed. If the remote response indicates that the URL leads to a further page of information at a remote web site then unless the newly received URL relates to a file which has already been done the file reference is added to the remote file queue to be processed subsequently at 545.

Once all of the remote URLs have been processed the thread closes, 550, and returns to 415 at Figure 4. At 416 the remote response codes collected at 530 and 535 of Figure 5 are processed to determine if there are errors so, for example, a remote file file not found indicator would result in an error being logged against the referring page information in the database.

Turning now to Figure 6, the HTTP object used to retrieve files is illustrated. The system is HTTP 1.1 compliant using connection keep-alives by default and handles chunked encoded responses. Thus, at 600, it the retrieved URL is not on the same server or port as previously the socket is closed otherwise the connection keep-alive enables a faster access. A check is carried on the buffers at 605 to determine whether or not the maximum capacity of the buffer is exceeded and if so the buffers are trimmed at 610 otherwise an outbound request header is built from the URL. If the connection has been kept alive then there is no requirement to reconnect to the server or proxy but if the socket is closed than at 625 connection to the server or appropriate proxy is made and the request built at step 615 is transmitted at 630 and the inbound response header is received at 635. A clear response indicates a HEAD request then no further action is required in respect of the HTTP fetch as determined at 640 and the process returns to the relevant point of Figure 3 for further processing. The header is now checked at 655 to determine whether the response is using chunked encoding and if not the whole of the body is read using the content length at 660. If chunked encoding is in use, then each chunk is read at 665 until a chunk ending with 0 indicating termination of the content is received at 670. Once the body content has been received it is determined whether the
request was sent via a proxy at 645 and if so the socket is closed at 650 prior to returning to step 315 of Figure 3.

Returning now to Figure 3, once a local file of mime type HTML has been recovered using HTTP fetch at step 310, at step 338 the page content is parsed and checked using the macro function of Figure 7 to which reference is now made. The function of the tests carried out in Figure 7 is to process the HTML content for the range of accessibility errors and to ensure that all of the links to frames, anchors, images and the like are added to appropriate file trees and work queues of the system to be subsequently checked. Thus at 700 a check is initially carried out to ensure that the string information matches and the page in then parsed for tags as shown in Figure 8 to which reference is also made. Each tag is stored as a set by the tag parser creating one set per tag name, each set being an array containing a tag object for each tag found containing start and end position of the tag and the tag attributes. Thus there will be frame tags, image tags, anchor tags and area tags through which the system cycles. Thus the system looks for the next opening on a page at 800 and assuming such an opening is found a check is carried out to determine if the opening is a comment start and if so a search is carried out for the comment close indicating that the tag is to text. Once the comment close is found the next opening is looked for and the system continues to cycle through looking at each opening in turn until a further opening in the page is not found.

In order to iterate through the tags in the order in which they are found, each tag object contains a pointer to the next tag such that even although each type of tag is in a different set the system iterates through tags in the order in which they are found. If a further opening is not found then the tag parser ends and returns the system to Figure 7. If an opening found is not a comment, then a search is carried out for a closing tag name, which if not found again causes the system to exit back to Figure 7. When a tag name is found at 815 a check is carried out to determine if the tag is simply for script at 820. Assuming that it is script again the end of the script is located and the system cycles to look for a further opening in the system. If at 820 it is found that the tag is not in script format then at step 830 the tag name is examined to see if it is of a type that requires further examination for accessibility thus enabling determination at 835 of whether the tag requires storing or further consideration by the checking program. If the tag is to be stored then a new tag object is created at 840 and stored in the appropriate tag set. if there are further attributes associated with the tag as indicated at 845 and if no further attributes are found the system again cycles to look for further openings. If an attribute associated with the tag is found then the value of the attribute is looked at 850 and if
found the attributes are stored at 855 or the attribute name is stored at 860 in the event of no attribute value having been associated. Thus when no further openings are found the system having created tag sets returns to Figure 7 where a search is carried out for a base tag. Assuming that at 710 a base tag is found there is no need to create one but if no such base tag exists one is created at 715. The system now works through all of the tags located checking for compliance with accessibility capabilities for example checking metatags at 720 for compliance & attributes of ALT functions at 725. Flicker instructions at 730 which may adversely effect readability style sheets at 735 and tables at 740 to ensure that headers and the like do not create alignment problems for table reading etc. are also checked. Links located with the page parsing function are added to the appropriate local remote queues and then for each tag type a file is added to the database using an add file process as indicated at 750 to 753. At the same time once an image file is added then a check is carried out on the ALT attributes associated with the image to ensure that they are present at 755. The tags having been checked can now be dropped from the tag parser as indicated at step 760 and the system returns as indicated on Figure 3 to determine whether there are more local files to be examined.

Returning then to Figures 1 and 2, step 180, once all of the local and remote files have been processed and checked, a report is created and, if an email address is present for the report function and an appropriate flag is set for sending an email report, then the results are formatted and an email sent. Although the engine may be used to send e-mail messages and reports, when the engine is running as a thread from the scheduler it is more effective for the scheduler to control the despatch of emails so that for example where multiple sites registered to a single publisher are being checked a composite report may be transmitted. Finally, the site information held in the database in respect of the latest run is stored at 195 along with statistical analysis results at step 200 prior to the engine spawned by the scheduler closing at 210.

In summary, the engine 6 actually performs the checking of the target web site 7. It runs as a standard Win32 process and is launched by the scheduler 4 whether the check is manually initiated or is a scheduled batch check. The engine 6 makes use of multiple threads and is written to be as fast as possible and use as few system and network resources as possible, thus enabling many checks to be run at once.

The engine 6 can return a range of response codes; these are used by the scheduler 4 to determine if a check was successful (and if it needs to be retried, if failed). The engine 6 works as follows:
The scheduler 4 starts up an engine process 6 passing a scheduled check id (see Fig 10).

The engine 6 takes the scheduled check id and looks up the start up parameters from the scheduled_checks table 50; it then also reads any extra parameters from the run_parameters.

The engine 6 then reads its proxy and domain configuration from the database and now cracks and checks the URL and other start up parameters. If the parameters are okay the engine 6 then begins the check by requesting a page, checking the HTML for accessibility errors and then adding all of the links to a queue of work to do.

The queue of local links is then worked through with accessibility errors being checked in the same manner, newly discovered links being added to appropriate queues. Once all local links have been investigated, the engine 6 starts up twenty threads to check all of the remote links. Using multiple threads speeds this process up especially when links are to Internet sites and proxies have to be used.

Once all checking is complete the engine 6 flushes the results to the database and performs some final sanity checks.

It will be noted that each target web-site being checked can only have one check running at one time. The Engine 6 always deletes/replaces results data for each run over a particular site URL, although it collects and saves historic run statistics for every run over a site.

The engine 6 performs as fast as possible and uses as few system resources as possible.

On HTTP connection "keep-alives" are used for all local requests. The chunked-encoding method for entity body transfer is understood.

Only local HTML files need to be fetched and parsed, hence HEAD requests are used for non-HTML and remote file types. If the returned file type is HTML despite having a non-HTML extension then it is re-requested using GET, so it is fully parsed. Smart Buffers are used internally in the HTTPGet class; this radically reduces page faults and heap resizing.

The HTTPGet class can optionally use its own internal heap separate from the main process heap. The justification for this is that this class allocates very large blocks which can disrupt the main heap and reduce locality-of-reference for other frequently referenced data items.

HEAD requests are used for efficiency – some web-sites/servers reject the standard HEAD request (possibly for security reasons) causing a communications failure
to be logged. This is a relatively rare occurrence which may be corrected for by using a further attempt using an alternative type of request on detection of such a failure.

Each page that is checked has its URL stored so it can be determined whether it has already been checked and if so what its response code was. An in memory directory structure is used to reduce space and look up time. The __fastcall calling convention is used, the whole process being compiled with speed optimisations on.

StringTables are used to avoid having to store duplicate strings with Standard Template Library (STL) being used for all strings, lists and vectors.

The scheduler is responsible for running manual and scheduled batch checks on sites and any other scheduled jobs/tasks. It runs as a Windows Service and as such is accessible through the standard windows services applet in control panel or administrative tasks on windows 2000. The scheduler is written in C++ and makes use of multiple threads to carry out tasks: namely setting the next run time of each scheduled check and scheduled job/task, starting and controlling every (immediate) manual check and each scheduled batch check and starting and controlling each scheduled task.

Apart from the database connection settings, which are fetched from the registry, all other configuration data is read from the database at scheduler start up. Once the scheduler has initialised itself it starts the above mentioned threads that monitor various tables looking for new work:

The SetRunTimes class reads from both the scheduled_checks and scheduled_jobs tables updating the next_run_time of any entry that requires this and has a valid run_times value. The run_times value for each scheduled check or job is in enhanced "crontab" format:

"mi hr dy mo dd yr" - (minute, hour, day of month, month, day of week(0-6), year)

with each field wildcardable or can be single, a list or a range. This value is used to specify the date and time of their possibly repeated cycle of automated runs.

The ScheduledChecks class implements the logic for running and managing checks; it looks at the scheduled_checks table and starts new checks when they are found with a current next_run_time. Once the scheduler has started a check it resumes looking for new checks and polling checks to see if they have ended. When a check ends the scheduler updates the corresponding scheduled_checks database entry. The scheduler can retry failed checks (which may be caused by a temporary condition), depending upon configuration entries.

The ScheduledTasks class implements the logic for running and managing jobs/tasks; it looks at the scheduled_jobs table and starts new tasks when they are found
with a current next_run_time. Once the scheduler has started a task it resumes looking for
new tasks and polling tasks to see if they have ended. When a task ends the scheduler
updates the corresponding scheduled_jobs database entry.

The scheduler uses the window registry to store a small amount of configuration
information required to connect to the database. All of the configuration data is stored
under the registry key for example,

HKEY_LOCAL_MACHINE\SOFTWARE\British Telecommunications Plc\

Each instance has its own sub-key where its data is stored; for example an
instance called BT_Retail would have its configuration data stored under the following
key.

HKEY_LOCAL..SOFTWARE\British Telecommunications Plc \ BT_Retail

The top-level key holds sub-keys for instances but could hold global configuration
for all instances.

In order to make it easy for all components to access the data in the registry
there is implemented a C++ class called Registry. This class can be found in the shared
code, there is also have a COM wrapper for this class.

The values stored in the registry for each instance are:

- Database DSN
- Database username
- Database password
- Virtual directory
- Executable location
- Log file location

The password for the database is stored in the registry in an encrypted form; to
further harden this you NTFS permissions may be used to protect the key. To do this set
the ACL would be set to only allow user accounts (the functional accounts various parts of
the system run under) access to the key.

The database will operate with either an MS Access 97 or a SQL Server (7 or
above) database. However, as previously mentioned, Access based systems would tend
to be small and not provide the required functionality of complex queries so that the
description hereinafter assumes an SQL Server.

Each instance has its own separate database (a separate SQL Server
installation is not necessary). The database is central to the system; it holds all site result
information as well as configuration data. The schema can be split into four parts:

- Tables that house the results data and run statistics for each checked site.
- Tables that house global standing data for all sites in the database, for example results types, results categories, mime types, HTTP response code descriptions etc.
- Tables that house configuration data, for example how many rows should the front-end reports display, how many concurrent checks should run etc.
- Tables that control the scheduling of checks and tasks.

There are several site results tables relating to a site. The sites table is the root table for a site and it is where the site_id resides along with the start URL for a check. Basically the sites table references one or more directories that in turn have one or more local_files. The local_files have one or more results. Each table relating to a site has been de-normalised to hold the site_id, this makes various queries much faster in the UI.

The engine and scheduler access the database through ODBC using two wrapper classes:
- DBCConnection
- DBRecordSet

The UI and the various house-keeping scripts use ADO.

The scheduler is a windows service that starts monitors and controls all the scheduler activities in the system based on entries in the database (Figure 10). It is registered or removed as a window service using a command line:
scheduler-instance=<instance>-install or

scheduler-instance=<instance>-remove

where <instance> names the website instance identifying its respective registry entries. The scheduler service is then started and stopped using the service management control under the windows control panel. Thus the scheduler has three main modes as shown in Figure 9 these being “install”, “remove” and “service”. Thus if an instance of scheduler is not registered as indicated at 901 the commend “install” from the window service command line will start the scheduler in stopped mode as indicated at 902. The scheduler reads registry entries for the <instance> at start up in each mode including route paths and database account details. The database details are used to connect to the respective database reading its full configuration etc when the service is started as indicated at 903.

The transition between 902 and 903 service stopped and service started is by means of a man machine interface command which starts or stops the service as appropriate. The other instance which will move service started to service stopped is the occurrence of fatal errors during program run.

Turning to Figure 10, the website database tables directly pertinent to the operation of the scheduler component are shown in an entity relationship diagram. The
principal tables are those of configuration 51 which contains all of the schedulers run time configuration apart from the instance identifying entries contained in the windows registry. Scheduled checks table 50 defines all of the manual and batch-type site checks which have run, are queued (manual entries) or are scheduled (batch entries) to run. The scheduled jobs table 52 defines all of the scheduled period or single shot jobs or tasks that have been or are due to run and email rules 53 defines when batched or immediate results and summary emails are sent and to whom after each check or batch of checks has been completed. Certain functions such as identity and other parameters are identified by PK or FK in brackets following the name of the data field where PK indicates a primary key and FK a foreign key for lining within the system. Identities may be standing data (indicated by (S)) or automatically generated during program run (indicated by (A)).

Thus turning to Figure 11, whilst referring also to Figure 9, the command line options mentioned above provide, for example, the instance name and install or remove instructions. These command line options are indicated at 1100 these coming in from the windows service command line for the installation and removal program shown in Figure 11 using the main functions CmdInstallService for registering the scheduler as a windows service, CmdRemoveService for deregistering scheduler service from windows, service_main which calls ServiceStart for starting and running the scheduler as a windows service ServiceStart calls the function ScheduledChecks::Run which controls the running of the website checks starting and handling the separate engine thread for each website's supplied URL. Thus at 1105 if the service is running already arguments will be present and an instance of the scheduler service is being started, whilst if an instance is being installed or removed the command line instances is used to get the registry configuration etc as indicated at 1110. In both instances, at 1115, the mode break out shows the potential instructions from the command line identifying the respective service functions to be run. In particular, if the mode is service then at 1120 StartService calls the main procedure which commences the service main function 1130 described hereinafter. The stop function is indicated by the remove service at 1125 and installing a new service at 1135.

Turning now to Figure 12, the first argument parsed from the service control manager is a service name containing an instance name of the form scheduler {<instance>}, <instance> being the name of this particular scheduler run instance. At 1205 the service instance is used to identify windows registry entries for accessing database and locating route and log directories etc. If the argument indicates a debug
service requirement then a flag will be set at 1220 prior to the registry data configurations being retrieved at 1215 together with the opening of a log file which may be named with a date stamp on opening. Basic configuration of the registry is also logged. The service handle is retrieved from the database at 1225 and if no such service handle is detected at 1230 then the service is stopped and a service stopped report is sent to the service control manager in windows at 1235. The service stopped function will also occur if the service fails to start after the report of “service start pending” to the service control manager at 1240. Assuming, at 1245, the service start has been successful the service function continues as required and start up of the threads performing core scheduler functionality with manual and automatic scheduled check threads running separate engine threads for each website being checked as required in accordance with the scheduled checks retrieved using the instance name. Service start returns if the service is shut down or if fatal error occurs to cause the scheduler service to exit via the report service stopped function at 1235. The start up service function 1250 is described in greater detail with reference to Figure 13.

Now referring to Figures 13 and 14, once the scheduler starts it will report 1300 its status to the service control manager and will recover the database configuration information (from Figure 10) at 1305. The configuration data is checked at 1310 and provided it is OK the scheduler begins to spawn separate concurrent processing threads for the automatic checks, manual checks, scheduled jobs and tasks and will set the next running times. Thus the database configuration is stored at 1315 and a check is carried out to determine whether scheduled batch checks are to be run at 1320. Scheduled checks are called for each instance of the automatic checks and manual checks, the automatic checks being run first in accordance with the schedule. Provided the spawning of automatic checks at 1325 is successful as determined at 1330, then if there are manual checks to be carried out as indicated by the configuration then manual checks threads are started to run, again in accordance with Figure 18. A check is carried out at 1345 to determine that the threads have been successfully spawned each thread running simultaneously in parallel and independently of the others returning to the current program on completion. The run type of a check is controlled by entries in the run types table 54 of Figure 10, the manual flag in the run type table 54 being either true for a manual check or false for an automatic check being started.

The scheduled checks spawned control, monitor and call website engine instances hereinbefore described, each check running as a new thread. Finally, a scheduled check thread determined from the scheduled jobs table 52 is spawned and
runs as hereinafter described with reference to figure 21. It will be noted that if any of the automatic check threads, manual check threads or scheduled task threads fail to start, then the appropriate email profile is checked at 1360 to determine whether an email message indicating failure should be sent to the appropriate email address dependent on the setting of various flags in respect of notifications required as indicated in email rules table 53. Thus the way in which email may be configured will be for emails to be sent to an email owner in accordance with the "email manager when" field of the email rules, these being for example on exit due to failure, on normal exit from scheduled programming, for neither of the above or both dependent upon the flag settings.

Returning them to Figure 14, if there is a requirement to email manager in respect of a failure, this will be carried at 1365 prior to writing information to a log and exiting back to the main program. If the database configuration fails at 1310 this also causes information to be deposited in a panic log for subsequent analysis and exit of the service at 1370.

Returning now to 1355, assuming that all of the checks are successfully running "set run time" threads are spawned at 1360, these threads calculating and setting next run time values for inclusion in table 50 for every scheduled check or job entry that has finished and has a valid run time value that is to say is to be run periodically. These values appear in tables 50 and 52 for all scheduled checks having a valid run time. Once again if a thread fails to run successfully at 1385 the system will exit as previously described with reference to failure of any of the other threads to operate. Once all of the necessary threads have been started the email flags of table 53 are checked to determine whether the owner or administrator should be notified of a successful start up of a website check and if necessary a start up email is sent at 1395. The main scheduler now waits for an event to be received from the running threads which would indicate a stop event from the service control manager or an exit of a child process for any reason. The wait at 1400 will be indefinite and any return from this point would indicate a requirement to close down the scheduler.

If a shut down occurs for any reason, any spawned threads which are still running are closed down at 1405 and a shut down event is logged. Again at 1410 the email rules table 53 is consulted to determine whether an email should be sent at 1415 indicating the scheduler has shut down, after which at 1420 the service is reported as having shut down to the service control manager.

Turning to Figures 15 to 17, the scheduler program is installed and removed manually from a DOS command window to provide a window service. Progress and
errors are displayed directly to the user with only limited details being written to the log file. As the program is operated in command mode it always exits regardless of success or failure. Thus, in installing the scheduler, with reference to Figure 15, the install command line of scheduler-install-instance=<instance> is provided to start the program at 1500.

The log file is updated at 1505 to show that the scheduler instance is being installed and, as indicated at 1510, the message installing scheduler service <instance> is displayed for the user in the man machine interface.

Module file name is retrieved at 1515 and a determination of the validity of the file name is made at 1520. Assuming that the module file name is valid the service control manager is opened at 1525 and successful opening is checked at 1530 enabling, at 1535, the creation of the service. A check to ensure that the service is properly installed is carried out at 1540 and provided this is so a screen output of <instance> installed is displayed at 1545. In the event of any failure at 1520, 1530 or 1540 an appropriate error message is displayed as indicated in Figure 15 at 1550, 1555 or 1560 as appropriate. Once the success or failure messages have been displayed to the user the service handle is closed in schedule service at 1565 and in the service control manager at 1570 and the program exits at 1580.

It will be noted that although the scheduler has been successfully installed it will not be running as a scheduled service until it is started from the windows service control manager subsequently.

In order to remove a scheduler instance, again from DOS command window, the program follows the steps of Figure 16 and 17 to which reference is now made. On receipt of the command line scheduler-remove-instance=<instance> the service starts at 1600 and records the removal of the scheduler instance to the log file at 1605 at the same time providing a screen output at 1610 indicating the removal of the scheduler service instance in question. The service control manager is opened at 1615 and assuming at 1620 it has successfully opened the service indicated is opened at 1625 and again provided that there is an indication at 1630 that the service opening has been successful the control service stop function is implemented at 1635. A time out will be started to enable the scheduler stop function to complete its course and assuming that the control command successfully transferred at 1640 the system initially waits for one second at 1645 to determine whether the scheduler instance has actually stopped. If the scheduler instance stops before expiry of the time out, as determined at 1650, the program will proceed to output an instance stopped message display at 1660. If, however, at 1650 the
service has not stopped the loop through 1645 and 1650 will check at one second intervals until the time out expires if the service fails to stop. It will be noted also that if the control service command does not successfully activate as determined at 1640 that the time out and sleep steps are omitted and at 1655 as the service has not stopped the screen output at 1685 indicates that the scheduler instance has failed to stop. Nevertheless, in either case an attempt is made to delete the service from the system at 1670 the service having actually stopped if the delete service succeeds.

Returning briefly to 1635, the control service command is issued regardless of whether a scheduler service is still running that is has been started in the service manager prior to receipt of the remove command line. If the service is already stopped then of course the system will immediately exit from the 1645, 1650 loop.

A check is carried out at 1675 to see that the service instance has been removed and if so a screen output indicating that the instance has been successfully removed is made at 1680. Alternatively, if the attempt to delete the service fails at 1685 an error message is displayed to the screen. Returning briefly to the attempts to open the service manager and open service at 1615 to 1630, if either of the service control manager or the service fail to open appropriate error messages are output at 1690 and 1700.

Finally, if the service has been removed the service handle is closed at 1705. The service control manager service handle is closed at 1710 which also occurs if the service has failed to open.

If the system has been unable to open the service control manager, as indicated at 1690, it simply exits at 1715 as will occur in the event of success.

Turning now to Figures 18 to 20, the auto-checks thread (spawned at 1325 of Figure 13 and the manual checks thread spawned at 1340 of Figure 13) calls the processing function as soon as each instance of scheduled checks class is started by the service start program of Figure 12.

The threads share the same code regardless of the instance of the scheduled checks class working out from the code by checking on whether an automatic or manual check is being carried. Thus the check type is read and logged at 1800 and the initialisation with a connection to the database and loading of configuration data is carried out at 1805. A check is carried out on the success or failure of the thread launch at 1810 since any errors during the initialisation stage are fatal and will result in a log error and incrementation of log error count at 1815 prior to returning to the service start program (Figure 13) which will cause the parent process also to shut down.
Returning now to Figure 18, at 1825 the threads logs on to the database with the account details read from the registry on start up of the scheduler. Provided that the log on is successful, as indicated at 1830, the emails rules from table 53 of Figure 10 are loaded at 1835 and a check of the successful loading of the email rules occurs at 1840. Results categories are also recovered from the database at 1850 from table 55 of Figure 10.

If at any point there is a failure then the error count is incremented and a serious error logged so that the system again returns to the service start program for shut down.

Now at 1860 it is determined whether the thread running is an auto-check or scheduled batch check and if so, at 1865, the status is updated in schedule check table 50 of Figure 10 to a configurable status. The status will be at start up if a previous scheduler was shut down causing an interrupt in the progress in the program for example by virtue of a return to the main program at 1820.

Again provided the scheduler update configures correctly, as indicated at 1870, other fields in the scheduled checks table 50 are updated and reset, for example by setting the next run time to a null value so that in the case of a scheduler having been shut down for an extended period any checks that should have been run during the extended shut down are stopped from running immediately. Next run time will be automatically recalculated to a next appropriate run time based on the current run time value.

The scheduled thread is now fully initialised, the start up phase of scheduled check instances having been completed and the long running phase of the scheduled check thread begins.

The steps run in an endless loop where all checks that are due to run and are new or queued result in the spawning of a website engine process and the system waits for each running check to finish and sends results emails as and when required and maintains the database updated. The scheduler will continue to run the loop until such time as a shut down from the service control manager or failure of the system for any reason.

A configurable time out is used while checking for a stop in each cycle. The loop pause value is configurable and can be reduced to 1mSec in circumstances where the scheduler is extremely busy. The loop pause, as used in 1890, is therefore reset at 1895 to the configurable value. Thus at 1900, where manual checks are to be run, a flag controlling reporting, the "all finished" flag is set to false where a manual check is to be run. This is indicated at 1905, while at 1910, where an automated check is to be run, the
“all finished” flag is set to true prior to reading details of all the checks due to run from the database. The sequence here selected from the scheduled checks tables of Figure 10 is in the abridged SQL format of

```
SELECT * FROM scheduled_checkssc, run_types rt WHERE (sc.status IN ('New', 'Retry', 'Queued')) OR sc.status IS NULL) AND sc.enabled = 1 AND sc.next_run_time < GETDATE() AND rt.id = sc.run_type_id AND manual = check_type> ORDER BY sc.priority DESC, sc.next_run_time ASC
```

The program now reads details of all of the checks which are due to run 1920, provided that they are of the same type as the first read check – that is manual checks are only run with other manual checks and auto checks are run as a single batch. The the details of all checks due to run in this instance of the scheduler are entered into a cursor, ensuring that all check details have successfully loaded 1925. If the operation is unsuccessful then the error count is incremented and the error written to the error log and this instance of the scheduler returns to the stop check at 1885 which determines whether the error count has exceeded a configurable value (in which case this instance of the scheduler program shuts down. Otherwise, the instance of scheduler sleeps for the configured time period 1890 prior to attempting a further run through the scheduled checks due.

Returning now to Figure 19, assuming that all of the check details have successfully loaded into the cursor, the cursor is incremented and the details of the next check to run is loaded 1935 and a determination made as to whether the entry in the cursor table is null or stop 1940 which would indicate that all scheduled checks have been run or are running. If all of the scheduled checks have not been started then the number of engine threads currently running is checked to determine whether the maximum number of threads (which is a configurable number) are currently already running 1845. If it is determined that the system is already at capacity then the system waits 1950 until at least one of the engine threads currently running completes its task before proceeding to the next stage. Otherwise, an engine thread is started with the details from the current check table 1955, the engine running as a parallel thread with other threads previously started either by the scheduler or by the other engine threads carrying out checks on remote websites. The start check 1955 handles error reports and the like from the running thread and updates the database in respect of any failures which may be encountered by the associated started thread.

Thus, referring additionally to figure 21, a check is carried out on the parameters received 2100 and if there are invalid parameters (for example NULL values) the engine thread
cannot be started which could be an indication of a malfunction within the scheduler so that the system necessarily returns to the main program after incrementing the error count and logging the error reason 2105.

If all of the parameters have been correctly received then the engine command line is constructed 2110 using the site name information and parameters prior to spawning the new engine thread 2115. The spawned engine will check a single website, the scheduled checks address or the URL of which is passed in the command line together with other parameters if required such as email reporting requirements. Some of the parameters which might be transferred are alternatively recovered directly by the engine process as hereinbefore described, directly from the tables mentioned in Figure 10.

Successful launch of the engine is checked at 2220 and if so (indicating that the instance of the engine is running for this website) then the parameters of the scheduled checks table are updated 2225 for example by updating the status to started and last run time to the current time to enable calculation of the next run time for example. The SQL sequence is then prepared for controlling the database update 2230 and the thread handle is closed at 2235 prior to transmitting the database update 2240 and checking that the update has been successful 2245. Assuming the check is successful then the process returns to figure 19 at 1960 otherwise the error count is incremented and the error type added to the log at 2105.

Returning then to 2220, if spawning of the engine fails for any reason the email parameters of figure 10 relating to the website under check are considered at 2250 to determine whether and to whom any reporting is required. If an email report is required then a failure report is generated and emailed 2255 otherwise the instructions for updating the scheduled checks table with status failed and the failure reason are constructed 2260 and the SQL sequence prepared 2265 to allow the database update and exit functions 2240, 2245 and 2105 to be completed.

Returning then to figure 19, assuming that the check has successfully started 1960 then the detail of the check currently being run is recorded (added) to a file and the thread count incremented 1970 so that the maximum thread count can be checked on return to load the next site to be checked. The program now pauses for a configurable period 1975 before returning to pick up the next site to be checked at 1935. Should the engine have failed to start then the fail count is incremented at 1965 and the next website address loaded without a pause.

Now if at 1935 the next entry in the cursor is null or indicates an end of file then at 1940 the system exits to 2000 where the "all finished" flag set at the start of the run of this
scheduler instance is in respect of a manual check or an automated (scheduled batch) check. If the check at 2000 sees the “all finished” flag set to true it will wait until all of the checks being run are complete 2005 so that any group mailing required can be carried out. If the scheduler instance being run is in respect of manual checks then the system waits for an engine reporting a check finished 2010 and then updates the database with final status and, subject to the emailing parameters sends a results email to the site owner or other designated recipient.

Provided that all checks are finished 2015, the email parameters of figure 10 are examined 2025 to determine whether a group emailing in respect of more than one website checked is required. If so, group (and site) emails are composed and sent as required 2030 prior to returning to step 1885 of figure 18 and sleeping until the next set of scheduled checks are required to be run.

If checks are still running at 2015 then the system may pause for other threads to finish 2020 in case group e-mail in respect of other manual checks may be needed.

Finally, if at 1885 a reason for the long-running scheduler to stop is found (because of an excess error count or a stop sequence from the DOS Command line then any running checks (engine threads) are closed down leaving the status flag at started in the scheduled checks table 2035, the log file is updated with the exit reason and the system returns to force the service start program to wake up from the non timeout sleep at 1400 of figure 14.

Although tag parsing and type sorting hereinbefore described has carried out with respect to currently specified checks for accessibility and accuracy, it will be appreciated that the tool may be adjusted to work with new accessibility requirements and additional types of tag without significant difficulty.

In final summary, the tool operates as a self-contained system, with manual checks initiated via the web UI and scheduled batch checks driven by database entries (which can be loaded from a CSV file generated within a text editor or spreadsheet). Checks are performed by accessing web sites exactly like a browser does, i.e. using the HTTP requests. It is best if the tool is connected to a network with direct access to the sites being checked – i.e. the intranet network for intranet sites. However, it is not essential for the tool to be so connected provided that password protected access to the sites being checked is available.

As previously mentioned, all registered intranet sites and their publisher details are preferably held and maintained using PD. PD should be arranged to export a CSV file of site URL and publisher email pairs which PD automatically ftp's onto the live internal
instance of the tool each day, into the specific FTP directory. For regular event checking, every seven days, just prior to the weekly batch check of all registered sites, the tool could run a scheduled task that loads the CSV file from PD into its database – inserting new entries, disabling old entries and updating existing entries in scheduled_checks. The tool will email publisher's if it is carrying out a weekly check and there is an associated email address in the PD export CSV file or it is carrying out a manual check and a valid email address is entered in the launch screen.

The tool has the facility for checking sites/pages secured using basic (plain text) authentication. There are no restrictions on who can access the main user interface so that anyone with access to the intranet can run manual checks. There are no restrictions on which site URL's can be checked (intranet or internet) but if necessary restrictions could be applied in future to prevent links to certain sites so that any attempt to link to such sites from a tested website will be flagged to the publisher or site owner so that inappropriate linking can be avoided.
CLAIMS

1. A website checking tool including processing means responsive to input of a website address to cause download of information from the page specified by the website address, the processing means
   Storing the website address,
   scanning the downloaded information,
   determining the content and format of the downloaded information and classifying elements found therein according to type,
   storing a respective list for a plurality of types of downloaded information and data associated therewith and,
   for each of the plurality of types of information testing each item in the respective list for compatibility with predetermined characteristics and/or operations,
   by determining from responses at each stage the compatibility of the information with the predetermined characteristics and/or operations and storing data identifying incompatibilities and associating said stored data with the input website address,
   at least one of said types being characterised by a being a URL or address of linked information, each such URL or address being stored in the corresponding list for its type and being checked for response when accessed, whereby incompatible information on a website identified by an input website address is identified.

2. A website checking tool as claimed in claim 1 characterised in that a further one of said types is an image and the processing means checks attributes associated therewith to identify potential incompatibility with accessibility requirements.

3. A website checking tool as claimed in claim 1 or claim 2 comprising a scheduler and a database which stores details of a plurality of websites to which the tool is to be applied, the scheduler maintaining a record of times at which each of the websites is checked and periodically determines from the database a list of websites to be checked and, in respect of each website in the list commences a search engine instance, the search engine instance causing a connection to a network, requesting download of a first page specified by a stored URL, parsing the information received to identify tags associated with features of the information, determining the type of information associated with each tag, checking the validity of any non-linking types of information against predetermined parameters, storing link data associated with linking information, separating...
the stored link data according to network and for each link data item associated with the same website as that specified by the stored URL repeating the process of downloading, parsing and validating until all linked pages within the same network have been checked and for each link data item not so tested checking the response received from other networks.

4. A website checking tool as claimed in claim 3 in which for each link data item associated with a different website the engine causes a request to be transmitted and checks the response, responses indicating error conditions causing storage of data associated with the location of the link data item.

5. A website checking tool as claimed in claim 4 in which the engine causes a plurality of link check threads to open, each said link check thread transmitting requests for respective information from URLs associated with respective link data items, each link check thread checking responses from the respective links and storing data in respect of error conditions encountered.

6. A website checking tool as claimed in claim 5 in which the link-check threads open gateways or proxy access servers to respective websites associated with particular URLs and retain said gateways or proxy access servers in an open condition while accessing further pages of information having URLs on the same website.

7. A website checking tool as claimed in any preceding claim running as a windows service program under control of a service control manager.

8. A website checking tool as claimed in any preceding claim and comprising at least one programmed computer or server including a scheduler program and an engine program, a database and an interface for access to a network and a user interface for receiving start and/or stop instructions and for displaying results of checks carried out.
Start

Command Line options

Instance name passed

Read configuration from registry

Has DSN?

Read run options from command line

Read run options from Database

Read Mime types & Domain & Proxies

Is Ready to run check

Fig 1

Exit

Fig 2
Fig 1
141
Create site entry in database
145
Look up Site Host Details
150
Add first file to Queue
155
Fetch File from Work Queue
160
Process Local File (Fig 3)
165
Is Local File Queue Empty?
170
Process Remote Files (Fig 4)
180
Post parse local files, logging files past the max. File size
185
Email Address & doing e-mail?
190
Send Results E-Mail
195
Store Site Info in Database
200
Store Process Statistics
210
Exit

Fig 2
Start

If server or port changed close socket

Smart Buffers over Maximum

Trim Buffers

Build Outbound Request Header

Socket Closed?

Connect to Server/Proxy

Send Request

Receive in-bound response header

Is HEAD Request?

Chunked Encoding?

Read Body using Content Length

Read Chunk

Request Via Proxy

Close Socket

End

6/22

Fig 6
Start

Check for string matches

Tag Parse the Page (Fig 8)

Base Tag?

Add Base Tag Attach to Current Page

Check META Tags

Check ALT Attributes

Check for Flicker

Check for Style Sheets

Check Tables

Add Links

For each Frame, Anchor, Area, Image

Add File

Add File

Add File

Add File

Check Image ALT Attributes

Drop Tags from Tag Parser

End

Fig 7
Not Registered

Install

remove

Service Stopped

MMC Start

MMC Stop

Service Started

Fatal errors

Fig 9
Fig 10
Start

Service start Arguments

Extract Service instance from argv[0]

Argv[1]= debug

yes

G_debug_mode=true

no

Get registry Data
Get Registry config
open Log File
log Registry Config

Get Service Handle
RegisterServiceCtrlHandler()

Report Service Start Pending to SCM

Service Handle?

Yes

OK?

No

Start-up Service Function (Fig 13/14)

Report Service stopped to SCM

Return

Fig 12

12/22
13/22

Start

Report Service up and running to SCM

Get Database Config

OK?

Yes

Log database Config

Auto Check Type

Spawn Auto Checks Thread (Fig 18)

No

OK?

Yes

Manual Check Type

Spawn Manual Checks Thread (Fig 18)

No

OK?

Yes

Spawn Scheduled Tasks Thread (Fig 21)

Fig 13
15/22

Start 1500

Log Installing Scheduler Instances 1505

Get Module File Name 1515

Unable to Install <instance> <error> 1550

OK? 1520

Yes

Open SC Manager 1525

No

Open SC manager Failed <error> 1555

OK? 1530

Yes

Create Service 1535

No

Create Service Failed <error> 1560

OK 1540

Yes

<instance> installed 1545

Close service Handle (Sch Service) 1565

Close Service Handle (sch SC Manager) 1570

Return 1580

Fig 15
Fig 22
# INTERNATIONAL SEARCH REPORT

## A. CLASSIFICATION OF SUBJECT MATTER

**IPC 7** 606F17/30

According to International Patent Classification (IPC) or to both national classification and IPC.

## B. FIELDS SEARCHED

**Minimum documentation searched** (classification system followed by classification symbols):

**IPC 7** 606F

**Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched**:

EPO-Internal, WPI Data, COMPENDEX, INSPEC

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>US 2002/156799 A1 (MARKEI STEPHEN ET AL) 24 October 2002 (2002-10-24) paragraphs 0003!/0011!, 0031!, 0050!-0052!, 0066!/0095!, 0100!/0109!, 0116!, claims 1, 2, 3, 37, 38, 63, figures</td>
<td>1-8</td>
</tr>
</tbody>
</table>

Further documents are listed in the continuation of box C. Patent family members are listed in annex.

* Special categories of cited documents:
  * A* document defining the general state of the art which is not considered to be of particular relevance
  * E* earlier document but published on or after the international filing date
  * L* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
  * O* document referring to an oral disclosure, use, exhibition or other means
  * P* document published prior to the international filing date but later than the priority date claimed

*Y* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

*X* document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

*Y* document of particular relevance: the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

*Z* document member of the same patent family

**Date of the actual completion of the International search**

24 May 2005

**Date of mailing of the International search report**

07/06/2005

**Name and mailing address of the ISA**

European Patent Office, P.B. 5816 Patentlaan 2 NL-2280 HV Rijswijk
Tel. (+31–70) 940–2040, Tlx. 31 651 epo nl, Fax (+31–70) 940–3016

**Authorized officer**

Emander, K
<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document, with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patent document cited in search report</td>
<td>Publication date</td>
<td>Patent family member(s)</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>US 2002156799 A1</td>
<td>24-10-2002</td>
<td>NONE</td>
</tr>
<tr>
<td>WO 03001413 A</td>
<td>03-01-2003</td>
<td>BR 0210589 A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 1410258 A1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 03001413 A1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2003126136 A1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2004230572 A1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2002147805 A1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 6470383 B1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 6541310 B1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 6549944 B1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 5958008 A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 5870559 A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 5974572 A</td>
</tr>
<tr>
<td>US 6601066 B1</td>
<td>29-07-2003</td>
<td>NONE</td>
</tr>
</tbody>
</table>