nology. While WDR **1100** contains field **1100***e*, field **1100***d* provides a standard and generic measurement for evaluating WDRs from different location technologies, without concern for the location technology used. The highest confidence entries to a WDR queue **22** are used regardless of which location technology contributed to the WDR queue **22**.

LBX Configuration

FIG. **12** depicts a flowchart for describing an embodiment of MS initialization processing. Depending on the MS, there are many embodiments of processing when the MS is powered on, started, restarted, rebooted, activated, enabled, or the like. FIG. **12** describes the blocks of processing relevant to the present disclosure as part of that initialization processing. It is recommended to first understand discussions of FIG. **19** for knowing threads involved, and variables thereof. Initialization processing starts at block **1202** and continues to block **1204** where the MS Basic Input Output System (BIOS) is initialized appropriately, then to block **1206** where other character **32** processing is initialized, and then to block **1208** to check if NTP is enabled for this MS. Block **1206** may start the preferred number of listen/receive threads for feeding queue **26** and the preferred number of send threads for sending data inserted to queue **24**, in particular when transmitting CK **1304** embedded in usual data **1302** and receiving CK **1304** or **1314** embedded in usual data **1302** or **1312**, respectively. The number of threads started should be optimal for parallel processing across applicable channel(s). In this case, other character **32** threads are appropriately altered for embedded CK processing (sending at first opportune outbound transmission; receiving in usual inbound transmission).

If block **1208** determines NTP is enabled (as defaulted or last set by a user (i.e. persistent variable)), then block **1210** initializes NTP appropriately and processing continues to block **1212**. If block **1208** determines NTP was not enabled, then processing continues to block **1212**. Block **1210** embodiments are well known in the art of NTP implementations (also see block **1626**). Block **1210** may cause the starting of thread(s) associated with NTP. In some embodiments, NTP use is assumed in the MS. In other embodiments, appropriate NTP use is not available to the MS. Depending on the NTP embodiment, thread(s) may pull time synchronization information, or may listen for and receive pushed time information. Resources **38** (or other MS local resource) provides interface to an MS clock for referencing, maintaining, and generating date/time stamps at the MS. After block **1210** processing, the MS clock is synchronized to NTP. Because of initialization of the MS in FIG. **12**, block **1210** may rely on a connected service to initially get the startup synchronized NTP date/time. MS NTP processing will ensure the NTP enabled/disabled variable is dynamically set as is appropriate (using semaphore access) because an MS may not have continuous clock source access during travel when needed for resynchronization. If the MS does not have access to a clock source when needed, the NTP use variable is disabled. When the MS has (or again gets) access to a needed clock source, then the NTP use variable is enabled.

Thereafter, block **1212** creates shared memory to maintain data shared between processes/threads, block **1214** initializes persistent data to shared memory, block **1216** initializes any non-persistent data to shared memory (e.g. some statistics **14**), block **1218** creates system queues, and block **1220** creates semaphore(s) used to ensure synchronous access by concurrent threads to data in shared memory, before continuing to block **1222**. Shared memory data accesses appropriately utilize semaphore lock windows (semaphore(s) created at block **1220**) for proper access. In one embodiment, block **1220** creates a single semaphore for all shared memory accesses, but this can deteriorate performance of threads accessing unrelated data. In the preferred embodiment, there is a semaphore for each reasonable set of data of shared memory so all threads are fully executing whenever possible. Persistent data is that data which maintains values during no power, for example as stored to persistent storage **60**. This may include data **8** (including permissions **10**, charters **12**, statistics **14**, service directory **16**), data **20**, LBX history **30**, data **36**, resources **38**, and/or other data. Persistent data preferably includes at least the DLMV (see DLM role(s) list Variable below), ILMV (see ILM role(s) list Variable below), process variables **19***xx*-Max values (**19***xx*=**1902**, **1912**, **1922**, **1932**, **1942** and **1952** (see FIG. **19** discussions below)) for the last configured maximum number of threads to run in the respective process, process variables **19***xx*-PID values (**19***xx*=**1902**, **1912**, **1922**, **1932**, **1942** and **1952** (see FIG. **19** discussions below)) for multi-purpose of: a) holding an Operating System Process Identifier (i.e. O/S PID) for a process started; and b) whether or not the respective process was last enabled (i.e. PID>0) or disabled (i.e. PID <=0), the confidence floor value (see FIG. **14**A), the WTV (see Whereabouts Timeliness Variable (see FIG. **14**A)), the NTP use variable (see FIG. **14**A) for whether or not NTP was last set to disabled or enabled (used at block **1208**), and the Source Periodicity Time Period (SPTP) value (see FIG. **14**B). There are reasonable defaults for each of the persistent data prior to the first use of MS **2** (e.g. NTP use is disabled, and only becomes enabled upon a successful enabling of NTP at least one time). Non-persistent data may include data involved in some regard to data **8** (and subsets of permissions **10**, charters **12**, statistics **14**, service directory **16**), data **20**, LBX history **30**, data **36**, resources **38**, queues, semaphores, etc. Block **1218** creates queues **22**, **24**, and **26**. Queues **1980** and **1990** are also created there if required. Queues **1980** and **1990** are not required when NTP is in use globally by participating data processing systems. Alternate embodiments may use less queues by threads sharing a queue and having a queue entry type field for directing the queue entry to the correct thread. Alternate embodiments may have additional queues for segregating entries of a queue disclosed for best possible performance. Other embodiments incorporate queues figuratively to facilitate explanation of interfaces between processing.

All queues disclosed herein are understood to have their own internally maintained semaphore for queue accesses so that queue insertion, peeking, accessing, etc uses the internally maintained semaphore to ensure two or more concurrently executing threads do not corrupt or misuse data to any queue. This is consistent with most operating system queue interfaces wherein a thread stays blocked (preempted) after requesting a queue entry until a queue entry appears in the queue. Also, no threads will collide with another thread when inserting, peeking, or otherwise accessing the same queue. Therefore, queues are implicitly semaphore protected. Other embodiments may use an explicit semaphore protected window around queue data accessing, in which case those semaphore(s) are created at block **1220**.

Thereafter, block **1222** checks for any ILM roles currently enabled for the MS (for example as determined from persistent storage of an ILM role(s) list Variable (ILMV) preferably preconfigured for the MS at first use, or configured as last configured by a user of the MS). ILM roles are maintained to the ILM role(s) list Variable (ILMV). The ILMV contains one or more entries for an ILM capability (role), each entry with a flag indicating whether it is enabled or disabled (marked=enabled, unmarked=disabled). If block **1222** deter-

mines there is at least one ILM role enabled (i.e. as marked by associated flag), then block **1224** artificially sets the corresponding **19***xx*-PID variables to a value greater than 0 for indicating the process(es) are enabled, and are to be started by subsequent FIG. **12** initialization processing. The **19***xx*-PID will be replaced with the correct Process Identifier (PID) upon exit from block **1232** after the process is started. Preferably, every MS can have ILM capability. However, a user may want to (configure) ensure a DLM has no ILM capability enabled (e.g. or having no list present). In some embodiments, by default, every MS has an unmarked list of ILM capability maintained to the ILMV for 1) USE DLM REFERENCES and 2) USE ILM REFERENCES. USE DLM REFERENCES, when enabled (marked) in the ILMV, indicates to allow the MS of FIG. **12** processing to determine its whereabouts relative remote DLMs. USE ILM REFERENCES, when enabled (marked) in the ILMV, indicates to allow the MS of FIG. **12** processing to determine its whereabouts relative remote ILMs. Having both list items marked indicates to allow determining MS whereabouts relative mixed DLMs and ILMs. An alternative embodiment may include a USE MIXED REFERENCES option for controlling the MS of FIG. **12** processing to determine its whereabouts relative mixed DLMs and/or ILMs. Alternative embodiments will enforce any subset of these options without exposing user configurations, for example on a MS without any means for being directly located.

For any of the ILMV roles of USE DLM REFERENCES, USE ILM REFERENCES, or both, all processes **1902**, **1912**, **1922**, **1932**, **1942** and **1952** are preferably started (i.e. **1902**-PID, **1912**-PID, **1922**-PID, **1932**-PID, **1942**-PID and **1952**-PID are artificially set at block **1224** to cause subsequent process startup at block **1232**). Characteristics of an anticipated LN-expanse (e.g. anticipated location technologies of participating MSs, MS capabilities, etc) will start a reasonable subset of those processes with at least process **1912** started. Block **1224** continues to block **1226**. If block **1222** determines there are no ILMV role(s) enabled, then block processing continues to block **1226**.

Block **1226** initializes an enumerated process name array for convenient processing reference of associated process specific variables described in FIG. **19**, and continues to block **1228** where the first member of the set is accessed for subsequent processing. The enumerated set of process names has a prescribed start order for MS architecture **1900**. Thereafter, if block **1230** determines the process identifier (i.e. **19***xx*-PID such that **19***xx* is **1902**, **1912**, **1922**, **1932**, **1942**, **1952** in a loop iteration of blocks **1228** through **1234**) is greater than 0 (e.g. this first iteration of **1952**-PID>0 implies it is to be started here; also implies process **1952** is enabled as used in FIGS. **14**A, **28**, **29**A and **29**B), then block **1232** spawns (starts) the process (e.g. **1952**) of FIG. **29**A to start execution of subordinate worker thread(s) (e.g. process **1952** thread(s)) and saves the real PID (Process Identifier) to the PID variable (e.g. **1952**-PID) returned by the operating system process spawn interface. Block **1232** passes as a parameter to the process of FIG. **29**A which process name to start (e.g. **1952**), and continues to block **1234**. If block **1230** determines the current process PID variable (e.g. **1952**-PID) is not greater than 0 (i.e. not to be started; also implies is disabled as used in FIGS. **14**A, **28**, **29**A and **29**B), then processing continues to block **1234**. Block **1234** checks if all process names of the enumerated set (pattern of **19***xx*) have been processed (iterated) by blocks **1228** through **1234**. If block **1234** determines that not all process names in the set have been processed (iterated), then processing continues back to block **1228** for handling the next process name in the set. If block **1234** determines that all process names of the enumerated set were processed, then block **1236** checks the DLMV (DLM role(s) list Variable). Blocks **1228** through **1234** iterate every process name of FIG. **19** to make sure that each is started in accordance with non-zero **19***xx*-PID variable values at FIG. **12** initialization.

Block **1236** checks for any DLM roles currently enabled for the MS (for example as determined from persistent storage of a DLM role(s) list Variable (DLMV) preferably preconfigured for the MS at first use if the MS contains DLM capability). DLM capability (roles), whether on-board at the MS, or determined during MS travels (see block **288**), is maintained to the DLM role(s) list Variable (DLMV). The DLMV contains one or more entries for a DLM capability (role), each (role) entry with a flag indicating whether it is enabled or disabled (marked=enabled, unmarked=disabled). If block **1236** determines there is at least one DLM role enabled (i.e. as marked by associated flag), then block **1238** initializes enabled role(s) appropriately and processing continues to block **1240**. Block **1238** may cause the starting of thread(s) associated with enabled DLM role(s), for DLM processing above (e.g. FIGS. **2**A through **9**B). Block **1238** may invoke API(s), enable flag(s), or initialize as is appropriate for DLM processing described above. Such initializations are well known in the art of prior art DLM capabilities described above. If block **1236** determines there are no DLM roles to initialize at the MS, then processing continues to block **1240**. Any of the FIG. **9**A technologies are eligible in the DLMV as determined to be present at the MS and/or as determined by historical contents of the WDR queue **22** (e.g. location technology field **1100***e* with MS ID field **1100***a* for this MS) and/or determined by LBX history **30**. Application Programming Interfaces (APIs) may also be used to determine MS DLM capability (role(s)) for entry(s) to the DLMV.

Block **1240** completes LBX character initialization, and FIG. **12** initialization processing terminates thereafter at block **1242**. Depending on what threads were started as part of block **1206**, Block **1240** may startup the preferred number of listen/receive threads for feeding queue **26** and the preferred number of send threads for sending data inserted to queue **24**, in particular when transmitting new data **1302** and receiving new data **1302** or **1312**. The number of threads started should be optimal for parallel processing across applicable channel(s). Upon encounter of block **1242**, the MS is appropriately operational, and a user at the MS of FIG. **12** processing will have the ability to use the MS and applicable user interfaces thereof.

With reference now to FIG. **29**A, depicted is a flowchart for describing a preferred embodiment of a process for starting a specified number of threads in a specified thread pool. FIG. **29**A is in itself an O/S process, has a process identifier (PID) after being started, will contain at least two threads of processing after being started, and is generic in being able to take on the identity of any process name passed to it (e.g. **19***xx*) with a parameter (e.g. from block **1232**). FIG. **29**A represents the parent thread of a **19***xx* process. The FIG. **29**A process is generic for executing any of processes **19***xx* (i.e. **1902**, **1912**, **1922**, **1932**, **1942** and **1952**) with the prescribed number of worker threads using the **19***xx*-Max configuration (i.e. **1902**-Max, **1912**-Max, **1922**-Max, **1932**-Max, **1942**-Max and **1952**-Max). FIG. **29**A will stay running until it (first all of its worker thread(s)) is terminated. FIG. **29**A consists of an O/S Process **19***xx* with at least a parent thread (main thread) and one worker thread (or number of worker threads for FIG. **19** processing as determined by **19***xx*-Max). The parent thread has purpose to stay running while all worker threads are running, and to own intelligence for starting worker threads and terminating the process when all worker threads are ter-

minated. The worker threads are started subordinate to the FIG. **29**A process at block **2912** using an O/S start thread interface.

A **19***xx* (i.e. **1902**, **1912**, **1922**, **1932**, **1942** and **1952**) process starts at block **2902** and continues to block **2904** where the parameter passed for which process name to start (i.e. take on identity of) is determined (e.g. **1952**). Thereafter, block **2906** creates a RAM semaphore (i.e. operating system term for a well performing Random Access Memory (RAM) semaphore with scope only within the process (i.e. to all threads of the process)). The local semaphore name preferably uses the process name prefix (e.g. **1952**-Sem), and is used to synchronize threads within the process. RAM semaphores perform significantly better than global system semaphores. Alternate embodiments will have process semaphore(s) created at block **1220** in advance. Thereafter, block **2908** initializes a thread counter (e.g. **1952**-Ct) to 0 for counting the number of worker threads actually started within the **19***xx* process (e.g. **1952**), block **2910** initializes a loop variable J to 0, and block **2912** starts a worker thread (the first one upon first encounter of block **2912** for a process) in this process (e.g. process **1902** starts worker thread FIG. **20**, . . . , process **1952** starts worker thread FIG. **26**A—see architecture **1900** description below).

Thereafter, block **2914** increments the loop variable by 1 and block **2916** checks if all prescribed worker threads have been started. Block **2916** accesses the **19***xx*-Max (e.g. **1952**-Max) variable from shared memory using a semaphore for determining the maximum number of threads to start in the process worker thread pool. If block **2916** determines all worker threads have been started, then processing continues to block **2918**. If block **2916** determines that not all worker threads have been started for the process of FIG. **29**A, then processing continues back to block **2912** for starting the next worker thread. Blocks **2912** through **2916** ensure the **19***xx*-Max (e.g. **1952**-Max) number of worker threads are started within the process of FIG. **29**A.

Block **2918** waits until all worker threads of blocks **2912** through **2916** have been started, as indicated by the worker threads themselves. Block **2918** waits until the process **19***xx*-Ct variable has been updated to the prescribed **19***xx*-Max value by the started worker threads, thereby indicating they are all up and running. When all worker threads are started (e.g. **1952**-Ct=**1952**-Max), thereafter block **2920** waits (perhaps a very long time) until the worker thread count (e.g. **1952**-Ct) has been reduced back down to 0 for indicating that all worker threads have been terminated, for example when the user gracefully powers off the MS. Block **2920** continues to block **2922** when all worker threads have been terminated. Block **2922** sets the shared memory variable for the **19***xx* process (e.g. **1952**-PID) to 0 using a semaphore for indicating that the **19***xx* (e.g. **1952**) process is disabled and no longer running. Thereafter, the **19***xx* process terminates at block **2924**. Waiting at blocks **2918** and **2920** are accomplished in a variety of well known methods:

- Detect signal sent to process by last started (or terminated) worker thread that thread count is now MAX (or 0); or
- Loop on checking the thread count with sleep time between checks, wherein within the loop there is a check of the current count (use RAM semaphore to access), and processing exits the loop (and block) when the count has reached the sought value; or
- Use of a semaphore for a count variable which causes the parent thread of FIG. **29**A to stay blocked prior to the count reaching its value, and causes the parent thread to become cleared (will leave wait block) when the count reaches its sought value.

Starting threads of processing in FIG. **29**A has been presented from a software perspective, but there are hardware/firmware thread embodiments which may be started appropriately to accomplish the same functionality. If the MS operating system does not have an interface for returning the PID at block **1232**, then FIG. **29**A can have a block (e.g. **2905**) used to determine its own PID for setting the **19***xx*-PID variable.

FIGS. **13**A through **13**C depict an illustration of data processing system wireless data transmissions over some wave spectrum. Embodiments may exist for any of the aforementioned wave spectrums, and data carried thereon may or may not be encrypted (e.g. encrypted WDR information). With reference now to FIG. **13**A, a MS, for example a DLM **200***a*, sends/broadcasts data such as a data **1302** in a manner well known to those skilled in the art, for example other character **32** processing data. When a Communications Key (CK) **1304** is embedded within data **1302**, data **1302** is considered usual communications data (e.g. protocol, voice, or any other data over conventional forward channel, reverse channel, voice data channel, data transmission channel, or any other prior art use channel) which has been altered to contain CK **1304**. Data **1302** contains a CK **1304** which can be detected, parsed, and processed when received by another MS or other data processing system in the vicinity of the MS (e.g. DLM **200***a*) as determined by the maximum range of transmission **1306**. CK **1304** permits "piggy-backing" on current transmissions to accomplish new functionality as disclosed herein. Transmission from the MS radiate out from it in all directions in a manner consistent with the wave spectrum used. The radius **1308** represents a first range of signal reception from the MS **200***a*, perhaps by another MS (not shown). The radius **1310** represents a second range of signal reception from the MS **200***a*, perhaps by another MS (not shown). The radius **1311** represents a third range of signal reception from the MS **200***a*, perhaps by another MS (not shown). The radius **1306** represents a last and maximum range of signal reception from the MS **200***a*, perhaps by another MS (not shown). MS design for maximum radius **1306** may take into account the desired maximum range versus acceptable wave spectrum exposure health risks for the user of the MS. The time of transmission from MS **200***a* to radius **1308** is less than times of transmission from MS **200***a* to radiuses **1310**, **1311**, or **1306**. The time of transmission from MS **200***a* to radius **1310** is less than times of transmission from MS **200***a* to radiuses **1311** or **1306**. The time of transmission from MS **200***a* to radius **1311** is less than time of transmission from MS **200***a* to radius **1306**.

In another embodiment, data **1302** contains a Communications Key (CK) **1304** because data **1302** is new transmitted data in accordance with the present disclosure. Data **1302** purpose is for carrying CK **1304** information for being detected, parsed, and processed when received by another MS or other data processing system in the vicinity of the MS (e.g. DLM **200***a*) as determined by the maximum range of transmission **1306**.

With reference now to FIG. **13**B, a MS, for example an ILM **1000***k*, sends/broadcasts data such as a data **1302** in a manner well known to those skilled in the art. Data **1302** and CK **1304** are as described above for FIG. **13**A. Data **1302** or CK **1304** can be detected, parsed, and processed when received by another MS or other data processing system in the vicinity of the MS (e.g. ILM **1000***k*) as determined by the maximum range of transmission **1306**. Transmission from the MS radiate out from it in all directions in a manner consistent with the wave spectrum used, and as described above for FIG. **13**A.

With reference now to FIG. **13**C, a service or set of services sends/broadcasts data such as a data packet **1312** in a manner well known to those skilled in the art, for example to service other character **32** processing. When a Communications Key (CK) **1314** is embedded within data **1312**, data **1312** is considered usual communications data (e.g. protocol, voice, or any other data over conventional forward channel, reverse channel, voice data channel, data transmission channel, or any other prior art use channel) which has been altered to contain CK **1314**. Data **1312** contains a CK **1314** which can be detected, parsed, and processed when received by an MS or other data processing system in the vicinity of the service(s) as determined by the maximum range of transmission **1316**. CK **1314** permits "piggy-backing" on current transmissions to accomplish new functionality as disclosed herein. Transmissions radiate out in all directions in a manner consistent with the wave spectrum used, and data carried thereon may or may not be encrypted (e.g. encrypted WDR information). The radius **1318** represents a first range of signal reception from the service (e.g. antenna thereof), perhaps by a MS (not shown). The radius **1320** represents a second range of signal reception from the service (e.g. antenna thereof), perhaps by a MS (not shown). The radius **1322** represents a third range of signal reception from the service (e.g. antenna thereof), perhaps by a MS (not shown). The radius **1316** represents a last and maximum range of signal reception from the service (e.g. antenna thereof), perhaps by a MS (not shown). The time of transmission from service to radius **1318** is less than times of transmission from service to radiuses **1320**, **1322**, or **1316**. The time of transmission from service to radius **1320** is less than times of transmission from service to radiuses **1322** or **1316**. The time of transmission from service to radius **1322** is less than time of transmission from service to radius **1316**. In another embodiment, data **1312** contains a Communications Key (CK) **1314** because data **1312** is new transmitted data in accordance with the present disclosure. Data **1312** purpose is for carrying CK **1314** information for being detected, parsed, and processed when received by another MS or data processing system in the vicinity of the service(s) as determined by the maximum range of transmission.

In some embodiments, data **1302** and **1312** are prior art wireless data transmission packets with the exception of embedding a detectable CK **1304** and/or CK **1314**, respectively. Usual data communications of MSs are altered to additionally contain the CK so data processing systems in the vicinity can detect, parse, and process the CK. Appropriate send and/or broadcast channel processing is used. In other embodiments, data **1302** and **1312** are new broadcast wireless data transmission packets for containing CK **1304** and CK **1314**, respectively. A MS may use send queue **24** for sending/broadcasting packets to data processing systems in the vicinity, and may use the receive queue **26** for receiving packets from other data processing systems in the vicinity. Contents of CKs (Communications Keys) depend on which LBX features are in use and the functionality intended.

In the case of "piggybacking" on usual communications, receive queue **26** insertion processing simply listens for the usual data and when detecting CK presence, inserts CK information appropriately to queue **26** for subsequent processing. Also in the case of "piggybacking" on usual communications, send queue **24** retrieval processing simply retrieves CK information from the queue and embeds it in an outgoing data **1302** at first opportunity. In the case of new data communications, receive queue **26** insertion processing simply listens for the new data containing CK information, and inserts CK information appropriately to queue **26** for subsequent processing. Also in the case of new data communications, send queue **24** retrieval processing simply retrieves CK information from the queue and transmits CK information as new data.

LBX: LN-EXPANSE Configuration

FIG. **14**A depicts a flowchart for describing a preferred embodiment of MS LBX configuration processing. FIG. **14** is of Self Management Processing code **18**. MS LBX configuration begins at block **1402** upon user action to start the user interface and continues to block **1404** where user interface objects are initialized for configurations described below with current settings that are reasonable for display to available user interface real estate. Thereafter, applicable settings are presented to the user at block **1406** with options. Block **1406** preferably presents to the user at least whether or not DLM capability is enabled (i.e. MS to behave as a DLM=at least one role of DLMV enabled), whether or not ILM capability is enabled (i.e. MS to behave as an ILM=at least one role of ILMV enabled), and/or whether or not this MS should participate in the LN-expanse as a source location for other MSs (e.g. process **1902** and/or **1942** enabled). Alternative embodiments will further present more or less information for each of the settings, or present information associated with other FIG. **14** blocks of processing. Other embodiments will not configure DLM settings for an MS lacking DLM capability (or when all DLMV roles disabled). Other embodiments will not configure ILM settings when DLM capability is present. Block **1406** continues to block **1408** where processing waits for user action in response to options. Block **1408** continues to block **1410** when a user action is detected. If block **1410** determines the user selected to configure DLM capability (i.e. DLMV role(s)), then the user configures DLM role(s) at block **1412** and processing continues back to block **1406**. Block **1412** processing is described by FIG. **15**A. If block **1410** determines the user did not select to configure DLM capability (i.e. DLMV role(s)), then processing continues to block **1414**. If block **1414** determines the user selected to configure ILM capability (i.e. ILMV role(s)), then the user configures ILM role(s) at block **1416** and processing continues back to block **1406**. Block **1416** processing is described by FIG. **15**B. If block **1414** determines the user did not select to configure ILM capability (i.e. ILMV role(s)), then processing continues to block **1418**. If block **1418** determines the user selected to configure NTP use, then the user configures NTP use at block **1420** and processing continues back to block **1406**. Block **1420** processing is described by FIG. **16**. If block **1418** determines the user did not select to configure NTP use, then processing continues to block **1422**.

If block **1422** determines the user selected to maintain the WDR queue, then the user maintains WDRs at block **1424** and processing continues back to block **1406**. Block **1424** processing is described by FIG. **17**. Blocks **1412**, **1416**, **1420** and **1424** are understood to be delimited by appropriate semaphore control to avoid multi-threaded access problems. If block **1422** determines the user did not select to maintain the WDR queue, then processing continues to block **1426**. If block **1426** determines the user selected to configure the confidence floor value, then block **1428** prepares parameters for invoking a Configure Value procedure (parameters for reference (address) of value to configure; and validity criteria of value to configure), and the Configure Value procedure of FIG. **18** is invoked at block **1430** with the two (2) parameters. Thereafter, processing continues back to block **1406**. Blocks **1428** and **1430** are understood to be delimited by appropriate semaphore control when modifying the confidence floor value since other threads can access the floor value.

The confidence floor value is the minimum acceptable confidence value of any field **1100***d* (for example as checked by block **276**). No WDR with a field **1100***d* less than the confidence floor value should be used to describe MS whereabouts. In an alternative embodiment, the confidence floor value is enforced as the same value across an LN-expanse with no user control to modify it. One embodiment of FIG. **14** does not permit user control over a minimum acceptable confidence floor value. Various embodiments will default the floor value. Block **1812** enforces an appropriate value in accordance with the confidence value range implemented (e.g. value from 1 to 100). Since the confidence of whereabouts is likely dependent on applications in use at the MS, the preferred embodiment is to permit user configuration of the acceptable whereabouts confidence for the MS. A new confidence floor value can be put to use at next thread(s) startup, or can be used instantly with the modification made, depending on the embodiment. The confidence floor value can be used to filter out WDRs prior to inserting to queue **22**, filter out WDRs when retrieving from queue **22**, filter out WDR information when listening on channel(s) prior to inserting to queue **26**, and/or used in accessing queue **22** for any reason (depending on embodiments). While confidence is validated on both inserts and queries (retrievals/peeks), one or the other validation is fine (preferably on inserts). It is preferred that executable code incorporate checks where applicable since the confidence floor value can be changed after queue **22** is in use. Also, various present disclosure embodiments may maintain all confidences to queue **22**, or a particular set of acceptable confidences.

If block **1426** determines the user did not select to configure the confidence floor value, then processing continues to block **1432**. If block **1432** determines the user selected to configure the Whereabouts Timeliness Variable (WTV), then block **1434** prepares parameters for invoking the Configure Value procedure (parameters for reference (address) of value to configure; and validity criteria of value to configure), and the Configure Value procedure of FIG. **18** is invoked at block **1430** with the two (2) parameters. Thereafter, processing continues back to block **1406**. Blocks **1434** and **1430** are understood to be delimited by appropriate semaphore control when modifying the WTV since other threads can access the WTV.

A critical configuration for MS whereabouts processing is whereabouts timeliness. Whereabouts timeliness is how often (how timely) an MS should have accurate whereabouts. Whereabouts timeliness is dependent on how often the MS is updated with whereabouts information, what technologies are available or are in the vicinity, how capable the MS is of maintaining whereabouts, processing speed(s), transmission speed(s), known MS or LN-expanse design constraints, and perhaps other factors. In some embodiments, whereabouts timeliness is as soon as possible. That is, MS whereabouts is updated whenever possible as often as possible. In fact, the present disclosure provides an excellent system and methodology to accomplish that by leveraging location technologies whenever and wherever possible. However, there should be balance when considering less capable processing of a MS to prevent hogging CPU cycles from other applications at the MS. In other embodiments, a hard-coded or preconfigured time interval is used for keeping an MS informed of its whereabouts in a timely manner. For example, the MS should know its own whereabouts at least every second, or at least every 5 seconds, or at least every minute, etc. Whereabouts timeliness is critical depending on the applications in use at the MS. For example, if MS whereabouts is updated once at the MS every 5 minutes during high speeds of travel when using navigation, the user has a high risk of missing a turn during travel in downtown cities where timely decisions for turns are required. On the other hand, if MS whereabouts is updated every 5 seconds, and an application only requires an update accuracy to once per minute, then the MS may be excessively processing.

In some embodiments, there is a Whereabouts Timeliness Variable (WTV) configured at the MS (blocks **1432**, **1434**, **1430**). Whether it is user configured, system configured, or preset in a system, the WTV is used to:

- Define the maximum period of time for MS whereabouts to become stale at any particular time;
- Cause the MS to seek its whereabouts if whereabouts information is not up to date in accordance with the WTV; and
- Prevent keeping the MS too busy with keeping abreast of its own whereabouts.

In another embodiment, the WTV is automatically adjusted based on successes or failures of automatically locating the MS. As the MS successfully maintains timely whereabouts, the WTV is maintained consistent with the user configured, system configured, or preset value, or in accordance with active applications in use at the time. However, as the MS fails in maintaining timely whereabouts, the WTV is automatically adjusted (e.g. to longer periods of time to prevent unnecessary wasting of power and/or CPU resources). Later, as whereabouts become readily available, the WTV can be automatically adjusted back to the optimal value. In an emergency situation, the user always has the ability to force the MS to determine its own whereabouts anyway (Blocks **856** and **862** through **878**, in light of a WDR request and WDR response described for architecture **1900**). In embodiments where the WTV is adjusted in accordance with applications in use at the time, the most demanding requirement of any application started is maintained to the WTV. Preferably, each application of the MS initializes to an API of the MS with a parameter of its WTV requirements. If the requirement is more timely than the current value, then the more timely value is used. The WTV can be put to use at next thread(s) startup, or can be used instantly with the modification made, depending on the embodiment.

If block **1432** determines the user did not select to configure the WTV, then processing continues to block **1436**. If block **1436** determines the user selected to configure the maximum number of threads in a **19***xx* process (see **19***xx*-Max variable in FIG. **19** discussions), then block **1438** interfaces with the user until a valid **19***xx*-max variable is selected, and processing continues to block **1440**. If block **1440** determines the **19***xx* process is already running (i.e. **19***xx*-PID>0 implies it is enabled), then an error is provided to the user at block **1442**, and processing continues back to block **1406**. Preferably, block **1442** does not continue back to block **1406** until the user acknowledges the error (e.g. with a user action). If block **1440** determines the user selected **19***xx* process (process **1902**, process **1912**, process **1922**, process **1932**, process **1942**, or process **1952**) is not already running (i.e. **19***xx*-PID=0 implies it is disabled), then block **1444** prepares parameters for invoking the Configure Value procedure (parameters for reference (address) of **19***xx*-Max value to configure; and validity criteria of value to configure), and the Configure Value procedure of FIG. **18** is invoked at block **1430** with the two (2) parameters. Thereafter, processing continues back to block **1406**. Blocks **1438**, **1440**, **1444** and **1430** are understood to be delimited by appropriate semaphore control when modifying the **19***xx*-Max value since other threads can access it. The **19***xx*-Max value should not be modified while the **19***xx* process is running because the number of threads to terminate may be changed prior to terminat-

ing. An alternate embodiment of modifying a process number of threads will dynamically modify the number of threads in anticipation of required processing.

If block **1436** determines the user did not select to configure a process thread maximum (**19***xx*-Max), then block **1446** checks if the user selected to (toggle) disable or enable a particular process (i.e. a **19***xx* process of FIG. **19**). If block **1446** determines the user did select to toggle enabling/disabling a particular FIG. **19** process, then block **1448** interfaces with the user until a valid **19***xx* process name is selected, and processing continues to block **1450**. If block **1450** determines the **19***xx* process is already running (i.e. **19***xx*-PID>0 implies it is enabled), then block **1454** prepares parameters (just as does block **2812**). Thereafter, block **1456** invokes FIG. **29**B processing (just as does block **2814**). Processing then continues back to block **1406**. If block **1450** determines the **19***xx* process is not running (i.e. **19***xx*-PID=0 implies it is disabled), then block **1452** invokes FIG. **29**A processing (just as does block **1232**). Processing then continues back to block **1406**. Block **1456** does not continue back to block **1406** until the process is completely terminated. Blocks **1448**, **1450**, **1452**, **1454** and **1456** are understood to be delimited by appropriate semaphore control.

Preferred embodiments of blocks **1446** and **1448** use convenient names of processes being started or terminated, rather than convenient brief process names such as **1902**, **1912**, **1922**, **1932**, **1942**, or **1952** used in flowcharts. In some embodiments, the long readable name is used, such as whereabouts broadcast process (**1902**), whereabouts collection process (**1912**), whereabouts supervisor process (**1922**), timing determination process (**1932**), WDR request process (**1942**), and whereabouts determination process (**1952**). For example, the user may know that the whereabouts supervisor process enabled/disabled indicates whether or not to have whereabouts timeliness monitored in real time. Enabling the whereabouts supervisor process enables monitoring for the WTV in real time, and disabling the whereabouts supervisor process disables monitoring the WTV in real time.

In another embodiment of blocks **1446** and **1448**, a completely new name or description may be provided to any of the processes to facilitate user interface usability. For example, a new name Peer Location Source Variable (PLSV) can be associated to the whereabouts broadcast process **1902** and/or **1942**. PLSV may be easier to remember. If the PLSV was toggled to disabled, the whereabouts broadcast process **1902** and/or **1942** terminates. If the PLSV was toggled to enabled, the whereabouts broadcast process **1902** and/or **1942** is started. It may be easier to remember that the PLSV enables/disables whether or not to allow this MS to be a location source for other MSs in an LN-expanse.

In other embodiments, a useful name (e.g. PLSV) represents starting and terminating any subset of **19***xx* processes (a plurality (e.g. **1902** and **1942**)) for simplicity. In yet other embodiments, FIG. **14**A/**14**B can be used to start or terminate worker thread(s) in any process, for example to throttle up more worker threads in a process, or to throttle down for less worker threads in a process, perhaps modifying thread instances to accommodate the number of channels for communications, or for the desired performance. There are many embodiments for fine tuning the architecture **1900** for optimal peer to peer interaction. In yet other embodiments, toggling may not be used. There may be individual options available at block **1408** for setting any data of this disclosure. Similarly, the **19***xx*-Max variables may be modified via individual user friendly names and/or as a group of **19***xx*-Max variables.

Referring back to block **1446**, if it is determined the user did not select to toggle for enabling/disabling process(es), then processing continues to block **1458**. If block **1458** determines the user selected to exit FIG. **14**A/**14**B configuration processing, then block **1460** terminates the user interface appropriately and processing terminates at block **1462**. If block **1458** determines the user did not select to exit the user interface, then processing continues to block **1466** of FIG. **14**B by way of off page connector **1464**.

With reference now to FIG. **14**B, depicted is a continued portion flowchart of FIG. **14**A for describing a preferred embodiment of MS LBX configuration processing. If block **1466** determines the user selected to configure the Source Periodicity Time Period (SPTP) value, then block **1468** prepares parameters for invoking the Configure Value procedure (parameters for reference (address) of value to configure; and validity criteria of value to configure), and the Configure Value procedure of FIG. **18** is invoked at block **1470** with the two (2) parameters. Thereafter, processing continues back to block **1406** by way of off page connector **1498**. Blocks **1468** and **1470** are understood to be delimited by appropriate semaphore control when modifying the SPTP value since other threads can access it. The SPTP configures the time period between broadcasts by thread(s) **1902**, for example 5 seconds. Some embodiments do not permit configuration of the SPTP.

If block **1466** determines the user did not select to configure the SPTP value, then processing continues to block **1472**. If block **1472** determines the user selected to configure service propagation, then the user configures service propagation at block **1474** and processing continues back to block **1406** by way of off page connector **1498**. If block **1472** determines the user did not select to configure service propagation, then processing continues to block **1476**.

If block **1476** determines the user selected to configure permissions **10**, then the user configures permissions at block **1478** and processing continues back to block **1406** by way of off page connector **1498**. If block **1476** determines the user did not select to configure permissions **10**, then processing continues to block **1480**. If block **1480** determines the user selected to configure charters **12**, then the user configures charters **12** at block **1482** and processing continues back to block **1406** by way of off page connector **1498**. If block **1480** determines the user did not select to configure charters **12**, then processing continues to block **1484**. If block **1484** determines the user selected to configure statistics **14**, then the user configures statistics **14** at block **1486** and processing continues back to block **1406** by way of off page connector **1498**. If block **1484** determines the user did not select to configure statistics **14**, then processing continues to block **1488**. If block **1488** determines the user selected to configure service informant code **28**, then the user configures code **28** at block **1490** and processing continues back to block **1406** by way of off page connector **1498**. If block **1488** determines the user did not select to configure code **28**, then processing continues to block **1492**. If block **1492** determines the user selected to maintain LBX history **30**, then the user maintains LBX history at block **1494** and processing continues back to block **1406** by way of off page connector **1498**. If block **1492** determines the user did not select to maintain LBX history **30**, then processing continues to block **1496**.

Block **1496** handles other user interface actions leaving block **1408**, and processing continues back to block **1406** by way of off page connector **1498**.

Details of blocks **1474**, **1478**, **1482**, **1486**, **1490**, **1494**, and perhaps more detail to block **1496**, are described with other flowcharts. Appropriate semaphores are requested at the beginning of block processing, and released at the end of block processing, for thread safe access to applicable data at

risk of being accessed by another thread of processing at the same time of configuration. In some embodiments, a user/administrator with secure privileges to the MS has ability to perform any subset of configurations of FIGS. **14**A and **14**B processing, while a general user may not. Any subset of FIG. **14** configuration may appear in alternative embodiments, with or without authenticated administrator access to perform configuration.

FIG. **15**A depicts a flowchart for describing a preferred embodiment of DLM role configuration processing of block **1412**. Processing begins at block **1502** and continues to block **1504** which accesses current DLMV settings before continuing to block **1506**. If there were no DLMV entries (list empty) as determined by block **1506**, then block **1508** provides an error to the user and processing terminates at block **1518**. The DLMV may be empty when the MS has no local DLM capability and there hasn't yet been any detected DLM capability, for example as evidenced by WDRs inserted to queue **22**. Preferably, the error presented at block **1508** requires the user to acknowledge the error (e.g. with a user action) before block **1508** continues to block **1518**. If block **1506** determines at least one entry (role) is present in the DLMV, then the current DLMV setting(s) are saved at block **1510**, the manage list processing procedure of FIG. **15**C is invoked at block **1512** with the DLMV as a reference (address) parameter, and processing continues to block **1514**.

Block **1514** determines if there were any changes to the DLMV from FIG. **15**C processing by comparing the DLMV after block **1512** with the DLMV saved at block **1510**. If there were changes via FIG. **15**C processing, such as a role which was enabled prior to block **1512** which is now disabled, or such as a role which was disabled prior to block **1512** which is now enabled, then block **1514** continues to block **1516** which handles the DLMV changes appropriately. Block **1516** continues to block **1518** which terminates FIG. **15**A processing. If block **1514** determines there were no changes via block **1512**, then processing terminates at block **1518**.

Block **1516** enables newly enabled role(s) as does block **1238** described for FIG. **12**. Block **1516** disables newly disabled role(s) as does block **2804** described for FIG. **28**.

FIG. **15**B depicts a flowchart for describing a preferred embodiment of ILM role configuration processing of block **1416**. Processing begins at block **1522** and continues to block **1524** which accesses current ILMV settings before continuing to block **1526**. If there were no ILMV entries (list empty) as determined by block **1526**, then block **1528** provides an error to the user and processing terminates at block **1538**. The ILMV may be empty when the MS is not meant to have ILM capability. Preferably, the error presented at block **1528** requires the user to acknowledge the error before block **1528** continues to block **1538**. If block **1526** determines at least one entry (role) is present in the ILMV, then the current ILMV setting(s) are saved at block **1530**, the manage list processing procedure of FIG. **15**C is invoked with a reference (address) parameter of the ILMV at block **1532**, and processing continues to block **1534**.

Block **1534** determines if there were any changes to the ILMV from FIG. **15**C processing by comparing the ILMV after block **1532** with the ILMV saved at block **1530**. If there were changes via FIG. **15**C processing, such as a role which was enabled prior to block **1532** which is now disabled, or such as a role which was disabled prior to block **1532** which is now enabled, then block **1534** continues to block **1536** which handles the ILMV changes appropriately. Block **1536** continues to block **1538** which terminates FIG. **15**B processing. If block **1534** determines there were no changes via block **1532**, then processing terminates at block **1538**.

Block **1536** enables newly enabled role(s) as does blocks **1224** through **1234** described for FIG. **12**. Block **1536** disables newly disabled role(s) as does blocks **2806** through **2816** described for FIG. **28**.

FIG. **15**C depicts a flowchart for describing a preferred embodiment of a procedure for Manage List processing. Processing starts at block **1552** and continues to block **1554**. Block **1554** presents the list (DLM capability if arrived to by way of FIG. **15**A; ILM capability if arrived to by way of FIG. **15**B) to the user, as passed to FIG. **15**C processing with the reference parameter by the invoker, with which list items are marked (enabled) and which are unmarked (disabled) along with options, before continuing to block **1556** for awaiting user action. Block **1554** highlights currently enabled roles, and ensures disabled roles are not highlighted in the presented list. When a user action is detected at block **1556**, thereafter, block **1558** checks if a list entry was enabled (marked) by the user, in which case block **1560** marks the list item as enabled, saves it to the list (e.g. DLMV or ILMV), and processing continues back to block **1554** to refresh the list interface. If block **1558** determines the user did not respond with an enable action, then block **1562** checks for a disable action. If block **1562** determines the user wanted to disable a list entry, then block **1564** marks (actually unmarks it) the list item as disabled, saves it to the list (e.g. DLMV or ILMV), and processing continues back to block **1554**. If block **1562** determines the user did not want to disable a list item, then block **1566** checks if the user wanted to exit FIG. **15**C processing. If block **1566** determines the user did not select to exit list processing, then processing continues to block **1568** where other user interface actions are appropriately handled and then processing continues back to block **1554**. If block **1566** determines the user did select to exit manage list processing, then FIG. **15**C processing appropriately returns to the caller at block **1570**.

FIG. **15**C interfaces with the user for desired DLMV (via FIG. **15**A) or ILMV (via FIG. **15**B) configurations. In some embodiments, it makes sense to have user control over enabling or disabling DLM and/or ILM capability (roles) to the MS, for example for software or hardware testing.

FIG. **16** depicts a flowchart for describing a preferred embodiment of NTP use configuration processing of block **1420**. Processing starts at block **1602** and continues to block **1604** where the current NTP use setting is accessed. Thereafter, block **1606** presents the current NTP use setting to its value of enabled or disabled along with options, before continuing to block **1608** for awaiting user action. When a user action is detected at block **1608**, block **1610** checks if the NTP use setting was disabled at block **1608**, in which case block **1612** terminates NTP use appropriately, block **1614** sets (and saves) the NTP use setting to disabled, and processing continues back to block **1606** to refresh the interface. Block **1612** disables NTP as does block **2828**.

If block **1610** determines the user did not respond for disabling NTP, then block **1616** checks for a toggle to being enabled. If block **1616** determines the user wanted to enable NTP use, then block **1618** accesses known NTP server address(es) (e.g. ip addresses preconfigured to the MS, or set with another user interface at the MS), and pings each one, if necessary, at block **1620** with a timeout. As soon as one NTP server is determined to be reachable, block **1620** continues to block **1622**. If no NTP server was reachable, then the timeout will have expired for each one tried at block **1620** for continuing to block **1622**. Block **1622** determines if at least one NTP server was reachable at block **1620**. If block **1622** determines no NTP server was reachable, then an error is presented to the user at block **1624** and processing continues back to

block **1606**. Preferably, the error presented at block **1624** requires the user to acknowledge the error before block **1624** continues to block **1606**. If block **1622** determines that at least one NTP server was reachable, then block **1626** initializes NTP use appropriately, block **1628** sets the NTP use setting to enabled (and saves), and processing continues back to block **1606**. Block **1626** enables NTP as does block **1210**.

Referring back to block **1616**, if it is determined the user did not want to enable NTP use, then processing continues to block **1630** where it is checked if the user wanted to exit FIG. **16** processing. If block **1630** determines the user did not select to exit FIG. **16** processing, then processing continues to block **1632** where other user interface actions leaving block **1608** are appropriately handled, and then processing continues back to block **1606**. If block **1630** determines the user did select to exit processing, then FIG. **16** processing terminates at block **1634**.

FIG. **17** depicts a flowchart for describing a preferred embodiment of WDR maintenance processing of block **1424**. Processing starts at block **1702** and continues to block **1704** where it is determined if there are any WDRs of queue **22**. If block **1704** determines there are no WDRs for processing, then block **1706** presents an error to the user and processing continues to block **1732** where FIG. **17** processing terminates. Preferably, the error presented at block **1706** requires the user to acknowledge the error before block **1706** continues to block **1732**. If block **1704** determines there is at least one WDR, then processing continues to block **1708** where the current contents of WDR queue **22** is appropriately presented to the user (in a scrollable list if necessary). Thereafter, block **1710** awaits user action. When a user action is detected at block **1710**, block **1712** checks if the user selected to delete a WDR from queue **22**, in which case block **1714** discards the selected WDR, and processing continues back to block **1708** for a refreshed presentation of queue **22**. If block **1712** determines the user did not select to delete a WDR, then block **1716** checks if the user selected to modify a WDR. If block **1716** determines the user wanted to modify a WDR of queue **22**, then block **1718** interfaces with the user for validated WDR changes before continuing back to block **1708**. If block **1716** determines the user did not select to modify a WDR, then block **1720** checks if the user selected to add a WDR to queue **22**. If block **1720** determines the user selected to add a WDR (for example, to manually configure MS whereabouts), then block **1722** interfaces with the user for a validated WDR to add to queue **22** before continuing back to block **1708**. If block **1720** determines the user did not select to add a WDR, then block **1724** checks if the user selected to view detailed contents of a WDR, perhaps because WDRs are presented in an abbreviated form at block **1708**. If it is determined at block **1724** the user did select to view details of a WDR, then block **1726** formats the WDR in detail form, presents it to the user, and waits for the user to exit the view of the WDR before continuing back to block **1708**. If block **1724** determines the user did not select to view a WDR in detail, then block **1728** checks if the user wanted to exit FIG. **17** processing. If block **1728** determines the user did not select to exit FIG. **17** processing, then processing continues to block **1730** where other user interface actions leaving block **1710** are appropriately handled, and then processing continues back to block **1708**. If block **1728** determines the user did select to exit processing, then FIG. **17** processing terminates at block **1732**.

There are many embodiments for maintaining WDRs of queue **22**. In some embodiments, FIG. **17** (i.e. block **1424**) processing is only provided for debug of an MS. In a single instance WDR embodiment, block **1708** presents the one and only WDR which is used to keep current MS whereabouts whenever possible. Other embodiments incorporate any subset of FIG. **17** processing.

FIG. **18** depicts a flowchart for describing a preferred embodiment of a procedure for variable configuration processing, namely the Configure Value procedure, for example for processing of block **1430**. Processing starts at block **1802** and continues to block **1804** where parameters passed by the invoker of FIG. **18** are determined, namely the reference (address) of the value for configuration to be modified, and the validity criteria for what makes the value valid. Passing the value by reference simply means that FIG. **18** has the ability to directly change the value, regardless of where it is located. In some embodiments, the parameter is an address to a memory location for the value. In another embodiment, the value is maintained in a database or some persistent storage, and FIG. **18** is passed enough information to know how to permanently affect/change the value.

Block **1804** continues to block **1806** where the current value passed is presented to the user (e.g. confidence floor value), and then to block **1808** for awaiting user action. When a user action is detected at block **1808**, block **1810** checks if the user selected to modify the value, in which case block **1812** interfaces with the user for a validated value using the validity criteria parameter before continuing back to block **1806**. Validity criteria may take the form of a value range, value type, set of allowable values, or any other criteria for what makes the value a valid one.

If block **1810** determines the user did not select to modify the value, then block **1814** checks if the user wanted to exit FIG. **18** processing. If block **1814** determines the user did not select to exit FIG. **18** processing, then processing continues to block **1816** where other user interface actions leaving block **1808** are appropriately handled, and then processing continues back to block **1806**. If block **1814** determines the user did select to exit processing, then FIG. **18** processing appropriately returns to the caller at block **1818**.

LBX: LN-EXPANSE Interoperability

FIG. **19** depicts an illustration for describing a preferred embodiment multithreaded architecture of peer interaction processing of a MS in accordance with the present disclosure. MS architecture **1900** preferably includes a set of Operating System (O/S) processes (i.e. O/S terminology "process" with O/S terminology "thread" or "threads (i.e. thread(s))), including a whereabouts broadcast process **1902**, a whereabouts collection process **1912**, a whereabouts supervisor process **1922**, a timing determination process **1932**, a WDR request process **1942**, and a whereabouts determination process **1952**. Further included are queues for interaction of processing, and process associated variables to facilitate processing. All of the FIG. **19** processes are of PIP code **6**. There is preferably a plurality (pool) of worker threads within each of said **19***xx* processes (i.e. **1902**, **1912**, **1922**, **1932**, **1942** and **1952**) for high performance asynchronous processing. Each **19***xx* process (i.e. **1902**, **1912**, **1922**, **1932**, **1942** and **1952**) preferably has at least two (2) threads:

1) "parent thread"; and

2) "worker thread".

A parent thread (FIG. **29**A) is the main process thread for:
- starting the particular process;
- starting the correct number of worker thread(s) of that particular process;
- staying alive while all worker threads are busy processing; and
- properly terminating the process when worker threads are terminated.

The parent thread is indeed the parent for governing behavior of threads at the process whole level. Every process has a name for convenient reference, such as the names **1902**, **1912**, **1922**, **1932**, **1942** and **1952**. Of course, these names may take on the associated human readable forms of whereabouts broadcast process, whereabouts collection process, whereabouts supervisor process, timing determination process, WDR request process, and whereabouts determination process, respectively. For brevity, the names used herein are by the process label of FIG. **19** in a form **19***xx*. There must be at least one worker thread in a process. Worker thread(s) are described with a flowchart as follows:

1902—FIG. **20**;
1912—FIG. **21**;
1922—FIG. **22**;
1932—FIG. **23**;
1942—FIG. **25**; and
1952—FIG. **26**A.

Threads of architecture MS are presented from a software perspective, but there are applicable hardware/firmware process thread embodiments accomplished for the same functionality. In fact, hardware/firmware embodiments are preferred when it is known that processing is mature (i.e. stable) to provide the fastest possible performance. Architecture **1900** processing is best achieved at the highest possible performance speeds for optimal wireless communications processing. There are two (2) types of processes for describing the types of worker threads:

1) "Slave to Queue"; and
2) "Slave to Timer".

A **19***xx* process is a slave to queue process when its worker thread(s) are driven by feeding from a queue of architecture **1900**. A slave to queue process stays "blocked" (O/S terminology "blocked"=preempted) on a queue entry retrieval interface until the sought queue item is inserted to the queue. The queue entry retrieval interface becomes "cleared" (O/S terminology "cleared"=clear to run) when the sought queue entry is retrieved from the queue by a thread. These terms (blocked and cleared) are analogous to a semaphore causing a thread to be blocked, and a thread to be cleared, as is well known in the art. Queues have semaphore control to ensure no more than one thread becomes clear at a time for a single queue entry retrieved (as done in an O/S). One thread sees a particular queue entry, but many threads can feed off the same queue to do the same work concurrently. Slave to queue type of processes are **1912**, **1932**, **1942** and **1952**. A slave to queue process is properly terminated by inserting a special termination queue entry for each worker thread to terminate itself after queue entry retrieval.

A **19***xx* process is a slave to timer process when its worker thread(s) are driven by a timer for peeking a queue of architecture **1900**. A timer provides the period of time for a worker thread to sleep during a looped iteration of checking a queue for a sought entry (without removing the entry from the queue). Slave to timer threads periodically peek a queue, and based on what is found, will process appropriately. A queue peek does not alter the peeked queue. The queue peek interface is semaphore protected for preventing peeking at an un-opportune time (e.g. while thread inserting or retrieving from queue). Queue interfaces ensure one thread is acting on a queue with a queue interface at any particular time. Slave to timer type of processes are **1902** and **1922**. A slave to timer process is properly terminated by inserting a special termination queue entry for each worker thread to terminate itself by queue entry peek.

Block **2812** knows the type of **19***xx* process for preparing the process type parameter for invocation of FIG. **29**B at block **2814**. The type of process has slightly different termination requirements because of the worker thread(s) processing type. Alternate embodiments of slave to timer processes will make them slave to queue processes by simply feeding off Thread Request (TR) queue **1980** for driving a worker thread when to execute (and when to terminate). New timer(s) would insert timely queue entries to queue **1980**, and processes **1902** and **1922** would retrieve from the queue (FIG. **24**A record **2400**). The queue entries would become available to queue **1980** when it is time for a particular worker thread to execute. Worker threads of processes **1902** and **1922** could retrieve, and stay blocked on, queue **1980** until an entry was inserted by a timer for enabling a worker thread (field **2400***a* set to **1902** or **1912**). TR queue **1980** is useful for starting any threads of architecture **1900** in a slave to queue manner. This may be a cleaner architecture for all thread pools to operate the same way (slave to queue). Nevertheless, the two thread pool methods are implemented.

Each **19***xx* process has at least four (4) variables for describing present disclosure processing:
- **19***xx*-PID=The O/S terminology "Process Identifier (PID)" for the O/S PID of the **19***xx* process. This variable is also used to determine if the process is enabled (PID>0), or is disabled (PID=0 (i.e. <=0));
- **19***xx*-Max=The configured number of worker thread(s) for the **19***xx* process;
- **19***xx*-Sem=A process local semaphore for synchronizing **19***xx* worker threads, for example in properly starting up worker threads in process **19***xx*, and for properly terminating worker threads in process **19***xx*; and
- **19***xx*-Ct=A process local count of the number of worker thread(s) currently running in the **19***xx* process.

19*xx*-PID and **19***xx*-Max are variables of PIP data **8**. **19***xx*-Sem and **19***xx*-Ct are preferably process **19***xx* stack variables within the context of PIP code **6**. **19***xx*-PID is a semaphore protected global variable in architecture **1900** so that it can be used to determine whether or not a particular **19***xx* process is enabled (i.e. running) or disabled (not running). **19***xx*-Max is a semaphore protected global variable in architecture **1900** so that user configuration processing outside of architecture **1900** can be used to administrate a desired number of worker threads for a **19***xx* process. Alternate embodiments will not provide user configuration of **19***xx*-Max variables (e.g. hard coded maximum number of threads), in which case no **19***xx*-Max global variable is necessary. "Thread(s) **19***xx*" is a brief form of stating "worker thread(s) of the **19***xx* process".

Receive (Rx) queue **26** is for receiving CK **1304** or CK **1314** data (e.g. WDR or WDR requests), for example from wireless transmissions. Queue **26** will receive at least WDR information (destined for threads **1912**) and WDR requests (FIG. **24**C records **2490** destined for threads **1942**). At least one thread (not shown) is responsible for listening on appropriate channel(s) and immediately depositing appropriate records to queue **26** so that they can be processed by architecture **1900**. Preferably, there is a plurality (pool) of threads for feeding queue **26** based on channel(s) being listened on, and data **1302** or **1312** anticipated for being received. Alternative embodiments of thread(s) **1912** may themselves directly be listening on appropriate channels and immediately

processing packets identified, in lieu of a queue **26**. Alternative embodiments of thread(s) **1942** may themselves directly be listening on appropriate channels and immediately processing packets identified, in lieu of a queue **26**. Queue **26** is preferred to isolate channel(s) (e.g. frequency(s)) and transmission reception processing in well known modular (e.g. Radio Frequency (RF)) componentry, while providing a high performance queue interface to other asynchronous threads of architecture **1900** (e.g. thread(s) of process **1912**). Wave spectrums (via particular communications interface **70**) are appropriately processed for feeding queue **26**. As soon as a record is received by an MS, it is assumed ready for processing at queue **26**. All queue **26** accesses are assumed to have appropriate semaphore control to ensure synchronous access by any thread at any particular time to prevent data corruption and misuse. Queue entries inserted to queue **26** may have arrived on different channel(s), and in such embodiments a channel qualifier may further direct queue entries from queue **26** to a particular thread **1912** or **1942** (e.g. thread(s) dedicated to channel(s)). In other embodiments, receive processing feeds queue **26** independent of any particular channel(s) monitored, or received on (the preferred embodiment described). Regardless of how data is received and then immediately placed on queue **26**, a received date/time stamp (e.g. fields **1100***p* or **2490***c*) is added to the applicable record for communicating the received date/time stamp to a thread (e.g. thread(s) **1912** or **1942**) of when the data was received. Therefore, the queue **26** insert interface tells the waiting thread(s) when the data was actually received. This ensures a most accurate received date/time stamp as close to receive processing as possible (e.g. enabling most accurate TDOA measurements). An alternate embodiment could determine applicable received date/time stamps in thread(s) **1912** or thread(s) **1942**. Other data placed into received WDRs are: wave spectrum and/or particular communications interface **70** of the channel received on, and heading/yaw/pitch/roll (or accelerometer readings) with AOA measurements, signal strength, and other field **1100***f* eligible data of the receiving MS. Depending on alternative embodiments, queue **26** may be viewed metaphorically for providing convenient grounds of explanation.

Send (Tx) queue **24** is for sending/communicating CK **1304** data, for example for wireless transmissions. At least one thread (not shown) is responsible for immediately transmitting (e.g. wirelessly) anything deposited to queue **24**. Preferably, there is a plurality (pool) of threads for feeding off of queue **24** based on channel(s) being transmitted on, and data **1302** anticipated for being sent. Alternative embodiments of thread(s) of processes **1902**, **1922**, **1932** and **1942** may themselves directly transmit (send/broadcast) on appropriate channels anything deposited to queue **24**, in lieu of a queue **24**. Queue **24** is preferred to isolate channel(s) (e.g. frequency(s)) and transmission processing in well known modular (e.g. RF) componentry, while providing a high performance queue interface to other asynchronous threads of architecture **1900** (e.g. thread(s) **1942**). Wave spectrums and/or particular communications interface **70** are appropriately processed for sending from queue **24**. All queue **24** accesses are assumed to have appropriate semaphore control to ensure synchronous access by any thread at any particular time to prevent data corruption and misuse. As soon as a record is inserted to queue **24**, it is assumed sent immediately. Preferably, fields sent depend on fields set. Queue entries inserted to queue **24** may contain specification for which channel(s) to send on in some embodiments. In other embodiments, send processing feeding from queue **24** has intelligence for which channel(s) to send on (the preferred embodiment described). Depending on alternative embodiments, queue **24** may be viewed metaphorically for providing convenient grounds of explanation.

When interfacing to queue **24**, the term "broadcast" refers to sending outgoing data in a manner for reaching as many MSs as possible (e.g. use all participating communications interfaces **70**), whereas the term "send" refers to targeting a particular MS or group of MSs.

WDR queue **22** preferably contains at least one WDR **1100** at any point in time, for at least describing whereabouts of the MS of architecture **1900**. Queue **22** accesses are assumed to have appropriate semaphore control to ensure synchronous access by any thread at any particular time to prevent data corruption and misuse. A single instance of data embodiment of queue **22** may require an explicit semaphore control for access. In a WDR plurality maintained to queue **22**, appropriate queue interfaces are again provided to ensure synchronous thread access (e.g. implicit semaphore control). Regardless, there is still a need for a queue **22** to maintain a plurality of WDRs from remote MSs. The preferred embodiment of all queue interfaces uses queue interface maintained semaphore(s) invisible to code making use of queue (e.g. API) interfaces. Depending on alternative embodiments, queue **22** may be viewed metaphorically for providing convenient grounds of explanation.

Thread Request (TR) queue **1980** is for requesting processing by either a timing determination (worker) thread of process **1932** (i.e. thread **1932**) or whereabouts determination (worker) thread of process **1952** (i.e. thread **1952**). When requesting processing by a thread **1932**, TR queue **1980** has requests (retrieved via processing **1934** after insertion processing **1918**) from a thread **1912** to initiate TDOA measurement. When requesting processing by a thread **1952**, TR queue **1980** has requests (retrieved via processing **1958** after insertion processing **1918** or **1930**) from a thread **1912** or **1922** so that thread **1952** performs whereabouts determination of the MS of architecture **1900**. Requests of queue **1980** comprise records **2400**. Preferably, there is a plurality (pool) of threads **1912** for feeding queue **1980** (i.e. feeding from queue **26**), and for feeding a plurality each of threads **1932** and **1952** from queue **1980**. All queue **1980** accesses are assumed to have appropriate semaphore control to ensure synchronous access by any thread at any particular time to prevent data corruption and misuse. Depending on alternative embodiments, queue **1980** may be viewed metaphorically for providing convenient grounds of explanation.

With reference now to FIG. **24**A, depicted is an illustration for describing a preferred embodiment of a thread request queue record, as maintained to Thread Request (TR) queue **1980**. TR queue **1980** is not required when a LN-expanse globally uses NTP, as found in thread **19***xx* processing described for architecture **1900**, however it may be required at a MS which does not have NTP, or a MS which interacts with another data processing system (e.g. MS) that does not have NTP. Therefore, TR queue record **2400** (i.e. queue entry **2400**) may, or may not, be required. This is the reason FIG. **1**A does not depict queue **1980**. When NTP is in use globally (in LN-expanse), TDOA measurements can be made using a single unidirectional data (**1302** or **1312**) packet containing a sent date/time stamp (of when the data was sent). Upon receipt, that sent date/time stamp received is compared with the date/time of receipt to determine the difference. The difference is a TDOA measurement. Knowing transmission speeds with a TDOA measurement allows calculating a distance. In this NTP scenario, no thread(s) **1932** are required.

Threads **1912** and/or DLM processing may always insert the MS whereabouts without requirement for thread(s) **1952**

by incorporating thread **1952** logic into thread **1912**, or by directly starting (without queue **1980**) a thread **1952** from a thread **1912**. Therefore, threads **1952** may not be required. If threads **1952** are not required, queue **1980** may not be required by incorporating thread **1932** logic into thread **1912**, or by directly starting (without queue **1980**) a thread **1932** from a thread **1912**. Therefore, queue **1980** may not be required, and threads **1932** may not be required.

Records **2400** (i.e. queue entries **2400**) contain a request type field **2400***a* and data field **2400***b*. Request type field **2400***a* simply routes the queue entry to destined thread(s) (e.g. thread(s) **1932** or thread(s) **1952**). A thread **1932** remains blocked on queue **1980** until a record **2400** is inserted which has a field **2400***a* containing the value **1932**. A thread **1952** remains blocked on queue **1980** until a record **2400** is inserted which has a field **2400***a* containing the value **1952**. Data field **2400***b* is set to zero (0) when type field **2400***a* contains **1952** (i.e. not relevant). Data field **2400***b* contains an MS ID (field **1100***a*) value, and possibly a targeted communications interface **70** (or wave spectrum if one to one), when type field contains **1932**. Field **2400***b* will contain information for appropriately targeting the MS ID with data (e.g. communications interface to use if MS has multiple of them). An MS with only one communications interface can store only a MS ID in field **2400***b*.

Records **2400** are used to cause appropriate processing by **19***xx* threads (e.g. **1932** or **1952**) as invoked when needed (e.g. by thread(s) **1912**). Process **1932** is a slave to queue type of process, and there are no queue **1980** entries **2400** which will not get timely processed by a thread **1932**. No interim pruning is necessary to queue **1980**.

With reference now back to FIG. **19**, Correlation Response (CR) queue **1990** is for receiving correlation data for correlating requests transmitted in data **1302** with responses received in data **1302** or **1312**. Records **2450** are inserted to queue **1990** (via processing **1928**) from thread(s) **1922** so that thread(s) **1912** (after processing **1920**) correlate data **1302** or **1312** with requests sent by thread(s) **1922** (e.g. over interface **1926**), for the purpose of calculating a TDOA measurement. Additionally, records **2450** are inserted to queue **1990** (via processing **1936**) from thread(s) **1932** so that thread(s) **1912** (after processing **1920**) correlate data **1302** or **1312** with requests sent by thread(s) **1932** (e.g. over interface **1938**), for the purpose of calculating a TDOA measurement. Preferably, there is a plurality (pool) of threads for feeding queue **1990** and for feeding from queue **1990** (feeding from queue **1990** with thread(s) **1912**). All queue **1990** accesses are assumed to have appropriate semaphore control to ensure synchronous access by any thread at any particular time to prevent data corruption and misuse. Depending on alternative embodiments, queue **1990** may be viewed metaphorically for providing convenient grounds of explanation.

With reference now to FIG. **24**B, depicted is an illustration for describing a preferred embodiment of a correlation response queue record, as maintained to Correlation Response (CR) queue **1990**. CR queue **1990** is not required when a LN-expanse globally uses NTP, as found in thread **19***xx* processing described for architecture **1900**, however it may be required at a MS which does not have NTP, or a MS which interacts with another data processing system (e.g. MS) that does not have NTP. Therefore, CR record **2450** (i.e. queue entry **2450**) may, or may not, be required. This is the reason FIG. **1**A does not depict queue **1990**. The purpose of CR queue **1990** is to enable calculation of TDOA measurements using correlation data to match a request with a response. When NTP is used globally in the LN-expanse, no such correlations between a request and response is required, as described above. In the NTP scenario, thread(s) **1912** can deduce TDOA measurements directly from responses (see FIG. **21**), and there is no requirement for threads **1932**.

TDOA measurements are best taken using date/time stamps as close to the processing points of sending and receiving as possible, otherwise critical regions of code may be required for enabling process time adjustments to the measurements when processing is "further out" from said points. This is the reason MS receive processing provides received date/time stamps with data inserted to queue **26** (field **1100***p* or **2490***c*). In a preferred embodiment, send queue **24** processing inserts to queue **1990** so the date/time stamp field **2450***a* for when sent is as close to just prior to having been sent as possible. However, there is still the requirement for processing time spent inserting to queue **1990** prior to sending anyway. Anticipated processing speeds of architecture **1900** allow reasonably moving sent date/time stamp setting just a little "further out" from actually sending to keep modular send processing isolated. A preferred embodiment (as presented) assumes the send queue **24** interface minimizes processing instructions from when data is placed onto queue **24** and when it is actually sent, so that the sending thread(s) **19***xx* (**1902**, **1922**, **1932** and **1942**) insert to queue **1990** with a reasonably accurate sent/date stamp field **2450***a*. This ensures a most accurate sent date/time stamp (e.g. enabling most accurate TDOA measurements). An alternate embodiment makes appropriate adjustments for more accurate time to consider processing instructions up to the point of sending after queue **1990** insertion.

Records **2450** (i.e. queue entries **2450**) contain a date/time stamp field **2450***a* and a correlation data field **2450***b*. Date/time stamp field **2450***a* contains a date/time stamp of when a request (data **1302**) was sent as set by the thread inserting the queue entry **2450**. Correlation data field **2450***b* contains unique correlation data (e.g. MS id with suffix of unique number) used to provide correlation for matching sent requests (data **1302**) with received responses (data **1302** or **1312**), regardless of the particular communications interface(s) used (e.g. different wave spectrums supported by MS). Upon a correlation match, a TDOA measurement is calculated using the time difference between field **2450***a* and a date/time stamp of when the response was received (e.g. field **1100***p*). A thread **1912** accesses queue **1990** for a record **2450** using correlation field **2450***b* to match, when data **1302** or **1312** contains correlation data for matching. A thread **1912** then uses the field **2450***a* to calculate a TDOA measurement. Process **1912** is not a slave to queue **1990** (but is to queue **26**). A thread **1912** peeks queue **1990** for a matching entry when appropriate. Queue **1990** may contain obsolete queue entries **2450** until pruning is performed. Some WDR requests may be broadcasts, therefore records **2450** may be used for correlating a plurality of responses. In another record **2450** embodiment, an additional field **2450***c* is provided for specification of which communication interface(s) and/or channel(s) to listen on for a response.

With reference now back to FIG. **19**, any reasonable subset of architecture **1900** processing may be incorporated in a MS. For example in one minimal subset embodiment, a DLM which has excellent direct locating means only needs a single instance WDR (queue **22**) and a single thread **1902** for broadcasting whereabouts data to facilitate whereabouts determination by other MSs. In a near superset embodiment, process **1942** processing may be incorporated completely into process **1912**, thereby eliminating processing **1942** by having threads **1912** feed from queue **26** for WDR requests as well as WDR information. In another subset embodiment, process **1922** may only send requests to queue **24** for responses, or

may only start a thread **1952** for determining whereabouts of the MS. There are many viable subset embodiments depending on the MS being a DLM or ILM, capabilities of the MS, LN-expanse deployment design choices, etc. A reference to FIG. **19** accompanies thread **19***xx* flowcharts (FIGS. **20**, **21**, **22**, **23**, **25** and **26**A). The user, preferably an administrator type (e.g. for lbxPhone™ debug) selectively configures whether or not to start or terminate a process (thread pool), and perhaps the number of threads to start in the pool (see FIG. **14**A). Starting a process (and threads) and terminating processes (and threads) is shown in flowcharts **29**A and **29**B. There are other embodiments for properly starting and terminating threads without departing from the spirit and scope of this disclosure.

LBX of data may also be viewed as LBX of objects, for example a WDR, WDR request, TDOA request, AOA request, charters, permissions, data record(s), or any other data may be viewed as an object. An subset of an object or data may also be viewed as an object.

FIG. **20** depicts a flowchart for describing a preferred embodiment of MS whereabouts broadcast processing, for example to facilitate other MSs in locating themselves in an LN-expanse. FIG. **20** processing describes a process **1902** worker thread, and is of PIP code **6**. Thread(s) **1902** purpose is for the MS of FIG. **20** processing (e.g. a first, or sending, MS) to periodically transmit whereabouts information to other MSs (e.g. at least a second, or receiving, MS) to use in locating themselves. It is recommended that validity criteria set at block **1444** for **1902**-Max be fixed at one (1) in the preferred embodiment. Multiple channels for broadcast at block **2016** should be isolated to modular send processing (feeding from a queue **24**).

In an alternative embodiment having multiple transmission channels visible to process **1902**, there can be a worker thread **1902** per channel to handle broadcasting on multiple channels. If thread(s) **1902** (block **2016**) do not transmit directly over the channel themselves, this embodiment would provide means for communicating the channel for broadcast to send processing when interfacing to queue **24** (e.g. incorporate a channel qualifier field with WDR inserted to queue **24**). This embodiment could allow specification of at least one (1) worker thread per channel, however multiple worker threads configurable for process **1902** as appropriated for the number of channels configurable for broadcast.

Processing begins at block **2002**, continues to block **2004** where the process worker thread count **1902**-Ct is accessed and incremented by 1 (using appropriate semaphore access (e.g. **1902**-Sem)), and continues to block **2006** for peeking WDR queue **22** for a special termination request entry. Block **2004** may also check the **1902**-Ct value, and signal the process **1902** parent thread that all worker threads are running when **1902**-Ct reaches **1902**-Max. Thereafter, if block **2008** determines that a worker thread termination request was not found in queue **22**, processing continues to block **2010**. Block **2010** peeks the WDR queue **22** (using interface **1904**) for the most recent highest confidence entry for this MS whereabouts by searching queue **22** for: the MS ID field **1100***a* matching the MS ID of FIG. **20** processing, and a confidence field **1100***d* greater than or equal to the confidence floor value, and a most recent NTP enabled date/time stamp field **1100***b* within a prescribed trailing period of time (e.g. preferably less than or equal to 2 seconds). For example, block **2010** peeks the queue (i.e. makes a copy for use if an entry found for subsequent processing, but does not remove the entry from queue) for a WDR of this MS (i.e. MS of FIG. **20** processing) which has the greatest confidence over 75 and has been most recently inserted to queue **22** with an NTP date/time stamp in the last 2 seconds. Date/time stamps for MS whereabouts which are not NTP derived have little use in the overall palette of process **19***xx* choices of architecture **1900** because receiving data processing systems (e.g. MSs) will have no means of determining an accurate TDOA measurement in the unidirectional transmission from an NTP disabled MS. A receiving data processing system will still require a bidirectional correlated exchange with the MS of FIG. **20** processing to determine an accurate TDOA measurement in its own time scale (which is accomplished with thread(s) **1922** pulling WDR information anyway). An alternate embodiment to block **2010** will not use the NTP indicator as a search criteria so that receiving data processing systems can receive to a thread **1912**, and then continue for appropriate correlation processing, or can at least maintain whereabouts to queue **22** to know who is nearby.

Thread **1902** is of less value to the LN-expanse when it broadcasts outdated/invalid whereabouts of the MS to facilitate locating other MSs. In an alternate embodiment, a movement tolerance (e.g. user configured or system set (e.g. 3 meters)) is incorporated at the MS, or at service(s) used to locate the MS, for knowing when the MS has significantly moved (e.g. more than 3 meters) and how long it has been (e.g. 45 seconds) since last significantly moving. In this embodiment, the MS is aware of the period of time since last significantly moving and the search time criteria is set using the amount of time since the MS significantly moved (whichever is greater). This way a large number of (perhaps more confident candidates) WDRs are searched in the time period when the MS has not significantly moved. Optional blocks **278** through **284** may have been incorporated to FIG. **2**F for movement tolerance processing just described, in which case the LWT is compared to the current date/time of block **2010** processing to adjust block **2010** search time criteria for the correct trailing period. In any case, a WDR is sought at block **2010** which will help other MSs in the LN-expanse locate themselves, and to let other MSs know who is nearby.

Thereafter, if block **2012** determines a useful WDR was found, then block **2014** prepares the WDR for send processing, block **2016** broadcasts the WDR information (using send interface **1906**) by inserting to queue **24** so that send processing broadcasts data **1302** (e.g. on all available communications interface(s) **70**), for example as far as radius **1306**, and processing continues to block **2018**. The broadcast is for reception by data processing systems (e.g. MSs) in the vicinity. At least fields **1100***b*, **1100***c*, **1100***d*, and **1100***n* are broadcast. See FIG. **11**A descriptions. Fields are set to the following upon exit from block **2014**:

MS ID field **1100***a* is preferably set with: Field **1100***a* from queue **22**, or transformed (if not already) into a pseudo MS ID (possibly for future correlation) if desired. This field may also be set to null (not set) because it is not required when the NTP indicator of field **1100***b* is enabled and the broadcast is sent with an NTP enabled field **1100***n*.

DATE/TIME STAMP field **1100***b* is preferably set with: Field **1100***b* from queue **22**.

LOCATION field **1100***c* is preferably set with: Field **1100***c* from queue **22**.

CONFIDENCE field **1100***d* is preferably set with: Field **1100***d* from queue **22**.

LOCATION TECHNOLOGY field **1100***e* is preferably set with: Field **1100***e* from queue **22**.

LOCATION REFERENCE INFO field **1100***f* is preferably set with: null (not set). Null indicates to send processing feeding from queue **24** to use all available comm. interfaces **70** (i.e. Broadcast). Specifying a comm. interface targets the specified interface (i.e. send).

COMMUNICATIONS REFERENCE INFO field **1100***g* is preferably set with: null (not set). If MS ID (or pseudo MS ID) is sent, this is all that is required to target this MS.

SPEED field **1100***h* is preferably set with: Field **1100***h* from queue **22**.

HEADING field **1100***i* is preferably set with: Field **1100***i* from queue **22**.

ELEVATION field **1100***j* is preferably set with: Field **1100***j* from queue **22**.

APPLICATION FIELDS field **1100***k* is preferably set with: Field **1100***k* from queue **22**. An alternate embodiment will add, alter, or discard data (with or without date/time stamps) here at the time of block **2014** processing.

CORRELATION FIELD **1100***m* is preferably set with: null (not set).

SENT DATE/TIME STAMP field **1100***n* is preferably set with: Sent date/time stamp as close in processing the broadcast of block **2016** as possible.

RECEIVED DATE/TIME STAMP field **1100***p* is preferably set with: Not Applicable (i.e. N/A for sending).

Block **2018** causes thread **1902** to sleep according to the SPTP setting (e.g. a few seconds). When the sleep time has elapsed, processing continues back to block **2006** for another loop iteration of blocks **2006** through **2016**. Referring back to block **2012**, if a useful WDR was not found (e.g. candidates too old), then processing continues to block **2018**. Referring back to block **2008**, if a worker thread termination request entry was found at queue **22**, then block **2020** decrements the worker thread count by 1 (using appropriate semaphore access (e.g. **1902**-Sem)), and thread **1902** processing terminates at block **2022**. Block **2020** may also check the **1902**-Ct value, and signal the process **1902** parent thread that all worker threads are terminated when **1902**-Ct equals zero (0).

Block **2016** causes broadcasting data **1302** containing CK **1304** wherein CK **1304** contains WDR information prepared as described above for block **2014**. Alternative embodiments of block **2010** may not search a specified confidence value, and broadcast the best entry available anyway so that listeners in the vicinity will decide what to do with it. A semaphore protected data access (instead of a queue peek) may be used in embodiments where there is always one WDR current entry maintained for the MS.

In the embodiment wherein usual MS communications data **1302** of the MS is altered to contain CK **1304** for listening MSs in the vicinity, send processing feeding from queue **24**, caused by block **2016** processing, will place WDR information as CK **1304** embedded in usual data **1302** at the next opportune time of sending usual data **1302**. If an opportune time is not timely, send processing should discard the send request of block **2016** to avoid broadcasting outdated whereabouts information (unless using a movement tolerance and time since last significant movement). As the MS conducts its normal communications, transmitted data **1302** contains new data CK **1304** to be ignored by receiving MS other character **32** processing, but to be found by listening MSs within the vicinity which anticipate presence of CK **1304**. Otherwise, when LN-Expanse deployments have not introduced CK **1304** to usual data **1302** communicated on a receivable signal by MSs in the vicinity, FIG. **20** sends repeated timely pulsed broadcasts of new data **1302** (per SPTP) for MSs in the vicinity of the first MS to receive. In any case, appropriate implementation should ensure field **1100***n* is as accurate as possible for when data **1302** is actually sent.

An alternate embodiment to architecture **1900** for elimination of process **1902** incorporates a trigger implementation for broadcasting MS whereabouts at the best possible time—i.e. when the MS whereabouts is inserted to queue **22**. As soon as a new (preferably NTP enabled) WDR candidate becomes available, it can be broadcast at a new block **279** of FIG. **2**F. (e.g. new block **279** continued to from block **278** and then continuing to block **280**). Fields are set as described above for FIG. **20**. Preferably, the new block **279** starts an asynchronous thread consisting of blocks **2014** and **2016** so that FIG. **2**F processing performance is not impacted. In a further embodiment, block **279** can be further enhanced using the SPTP value to make sure that too many broadcasts are not made. The SPTP (Source Periodicity Time Period) could be observed for getting as close as possible to broadcasting whereabouts in accordance with SPTP (e.g. worst case there are not enough broadcasts).

FIG. **21** depicts a flowchart for describing a preferred embodiment of MS whereabouts collection processing. FIG. **21** processing describes a process **1912** worker thread, and is of PIP code **6**. Thread(s) **1912** purpose is for the MS of FIG. **21** processing (e.g. a second, or receiving, MS) to collect potentially useful WDR information from other MSs (e.g. at least a first, or sending, MS) in the vicinity for determining whereabouts of the receiving (second) MS. It is recommended that validity criteria set at block **1444** for **1912**-Max be set as high as possible (e.g. 10) relative performance considerations of architecture **1900**, with at least one thread per channel that WDR information may be received on by the receiving MS. Multiple channels for receiving data fed to queue **26** should be isolated to modular receive processing (feeding a queue **26**).

In an alternative embodiment having multiple receiving transmission channels visible to process **1912** (e.g. thread(s) **1912** receiving directly), there can be a worker thread **1912** per channel to handle receiving on multiple channels simultaneously. If thread(s) **1912** do not receive directly from the channel, the preferred embodiment of FIG. **21** would not need to convey channel information to thread(s) **1912** waiting on queue **26** anyway. Embodiments could allow specification/configuration of many thread(s) **1912** per channel.

Processing begins at block **2102**, continues to block **2104** where the process worker thread count **1912**-Ct is accessed and incremented by 1 (using appropriate semaphore access (e.g. **1912**-Sem)), and continues to block **2106** for interim housekeeping of pruning the WDR queue by invoking a Prune Queues procedure of FIG. **27**. Block **2104** may also check the **1912**-Ct value, and signal the process **1912** parent thread that all worker threads are running when **1912**-Ct reaches **1912**-Max. Block **2106** may not be required since block **2130** can cause queue **22** pruning (block **292**).

Thereafter, block **2108** retrieves from queue **26** a WDR (using interface **1914**), perhaps a special termination request entry, or a WDR received in data **1302** (CK **1304**) or data **1312** (CK **1314**), and only continues to block **2110** when a WDR has been retrieved. Block **2108** stays blocked on retrieving from queue **26** until any WDR is retrieved. If block **2110** determines that a special WDR indicating to terminate was not found in queue **26**, processing continues to block **2112**. Block **2112** adjusts date/time stamp field **1100***b* if necessary depending on NTP use in the LN-expanse and adjusts the confidence field **1100***d* accordingly. In a preferred embodiment, fields **1100***b* and **1100***d* for the WDR in process is set as follows for certain conditions:

- Fields **1100***b*, **1100***n* and **1100***p* all NTP indicated: keep fields **1100***b* and **1100***d* as is; or
- Fields **1100***b* and **1100***n* are NTP indicated, **1100***p* is not: Is correlation (field **1100***m*) present?: No, then set confidence (field **1100***d*) to 0 (for filtering out at block **2114**)/Yes, then set field **1100***b* to **1100***p* (in time terms of this

MS) and adjust confidence lower based on differences between fields **1100***b*, **1100***n* and **1100***p*; or

Fields **1100***b* and **1100***p* are NTP indicated, **1100***n* is not: Is correlation present?: No, then set confidence to 0 (for filtering out at block **2114**)/Yes, then set field **1100***b* to **1100***p* (in time terms of this MS) and adjust confidence lower based on differences between fields **1100***b*, **1100***n* and **1100***p*; or

Fields **1100***b* NTP indicated, **1100***n* and **1100***p* not: Is correlation present?: No, then set confidence to 0 (for filtering out at block **2114**)/Yes, then set field **1100***b* to **1100***p* (in time terms of this MS) and adjust confidence lower based on differences between fields **1100***b*, **1100***n* and **1100***p*; or

Field **1100***b* not NTP indicated, **1100***n* and **1100***p* are: Is correlation present?: No, then set confidence to 0 (for filtering out at block **2114**)/Yes, then set field **1100***b* to **1100***p* (in time terms of this MS) and adjust confidence lower based on differences between fields **1100***b*, **1100***n* and **1100***p*; or

Fields **1100***b* and **1100***p* are not NTP indicated, **1100***n* is: Is correlation present?: No, then set confidence to 0 (for filtering out at block **2114**)/Yes, then set field **1100***b* to **1100***p* (in time terms of this MS) and adjust confidence lower based on differences between fields **1100***b*, **1100***n* and **1100***p*; or

Fields **1100***b* and **1100***n* are not NTP indicated, **1100***p* is: Is correlation present?: No, then set confidence to 0 (for filtering out at block **2114**)/Yes, then set field **1100***b* to **1100***p* (in time terms of this MS) and adjust confidence lower based on differences between fields **1100***b*, **1100***n* and **1100***p*; or

Fields **1100***b*, **1100***n* and **1100***p* not NTP indicated: Is correlation present?: No, then set confidence to 0 (for filtering out at block **2114**)/Yes, then set field **1100***b* to **1100***p* (in time terms of this MS) and adjust confidence lower based on differences between fields **1100***b*, **1100***n* and **1100***p*.

NTP ensures maintaining a high confidence in the LN-expanse, but absence of NTP is still useful. Confidence values should be adjusted with the knowledge of the trailing time periods used for searches when sharing whereabouts (e.g. thread(s) **1942** searches). Block **2112** continues to block **2114**.

If at block **2114**, the WDR confidence field **1100***d* is not greater than the confidence floor value, then processing continues back to block **2106**. If block **2114** determines that the WDR field **1100***d* is satisfactory, then block **2116** initializes a TDOA_FINAL variable to False, and block **2118** checks if the WDR from block **2108** contains correlation (field **1100***m*).

If block **2118** determines the WDR does not contain correlation, then block **2120** accesses the ILMV, block **2122** determines the source (ILM or DLM) of the WDR using the originator indicator of field **1100***e*, and block **2124** checks suitability for collection of the WDR. While processes **19***xx* running are generally reflective of the ILMV roles configured, it is possible that the more descriptive nature of ILMV role(s) not be one to one in relationship to **19***xx* processes, in particular depending on the subset of architecture **1900** in use. Block **2124** is redundant anyway because of block **274**. If block **2124** determines the ILMV role is disabled for collecting this WDR, then processing continues back to block **2106**. If block **2124** determines the ILMV role is enabled for collecting this WDR, then processing continues to block **2126**.

If block **2126** determines both the first (sending) and second (receiving) MS are NTP enabled (i.e. Fields **1100***b*, **1100***n* and **1100***p* are NTP indicated) OR if TDOA_FINAL is set to True (as arrived to via block **2150**), then block **2128** completes the WDR for queue **22** insertion, block **2130** prepares parameters for FIG. **2**F processing and block **2132** invokes FIG. **2**F processing (interface **1916**). Parameters set at block **2130** are: WDRREF=a reference or pointer to the WDR completed at block **2128**; DELETEQ=FIG. **21** location queue discard processing; and SUPER=FIG. **21** supervisory notification processing. Block **2128** calculates a TDOA measurement whenever possible and inserts to field **1100***f*. See FIG. **11**A descriptions. Fields are set to the following upon exit from block **2128**:

MS ID field **1100***a* is preferably set with: Field **1100***a* from queue **26**.

DATE/TIME STAMP field **1100***b* is preferably set with: Preferred embodiment discussed for block **2112**.

LOCATION field **1100***c* is preferably set with: Field **1100***c* from queue **26**.

CONFIDENCE field **1100***d* is preferably set with: Confidence at equal to or less than field **1100***d* received from queue **26** (see preferred embodiment for block **2112**).

LOCATION TECHNOLOGY field **1100***e* is preferably set with: Field **1100***e* from queue **26**.

LOCATION REFERENCE INFO field **1100***f* is preferably set with: All available measurements from receive processing (e.g. AOA, heading, yaw, pitch, roll, signal strength, wave spectrum, particular communications interface **70**, etc), and TDOA measurement(s) as determined in FIG. **21** (blocks **2128** and **2148**).

COMMUNICATIONS REFERENCE INFO field **1100***g* is preferably set with: Field **1100***g* from queue **26**.

SPEED field **1100***h* is preferably set with: Field **1100***h* from queue **26**.

HEADING field **1100***i* is preferably set with: Field **1100***i* from queue **26**.

ELEVATION field **1100***j* is preferably set with: Field **1100***j* from queue **26**.

APPLICATION FIELDS field **1100***k* is preferably set with: Field **1100***k* from queue **26**. An alternate embodiment will add, alter, or discard data (with or without date/time stamps) here at the time of block **2128** processing.

CORRELATION FIELD **1100***m* is preferably set with: Not Applicable (i.e. not maintained to queue **22**). Was used by FIG. **21** processing.

SENT DATE/TIME STAMP field **1100***n* is preferably set with: Not Applicable (i.e. not maintained to queue **22**). Was used by FIG. **21** processing.

RECEIVED DATE/TIME STAMP field **1100***p* is preferably set with: Not Applicable (i.e. not maintained to queue **22**). Was used by FIG. **21** processing.

Block **2132** continues to block **2134** where a record **2400** is built (i.e. field **2400***a*=**1952** and field **2400***b* is set to null (e.g. −1)) and then block **2136** inserts the record **2400** to TR queue **1980** (using interface **1918**) so that a thread **1952** will perform processing. Blocks **2134** and **2136** may be replaced with an alternative embodiment for starting a thread **1952**. Block **2136** continues back to block **2106**.

Referring now back to block **2126**, if it is determined that a TDOA measurement cannot be made (i.e. (field **1100***n* or **1100***p* not NTP indicated) OR if TDOA_FINAL is set to False), then block **2138** checks if the WDR contains a MS ID (or pseudo MS ID). If block **2138** determines there is none, then processing continues back to block **2106** because there is no way to distinguish one MS from another with respect to the WDR retrieved at block **2108** for directing bidirectional correlation. An alternate embodiment will use a provided correlation field **1100***m* received at block **2108**, instead of a field **1100***a*, for knowing how to target the originating MS for

TDOA measurement processing initiated by a thread **1932**. If block **2138** determines there is a usable MS ID (or correlation field), then block **2140** builds a record **2400** (field **2400***a*=**1932**, field **2400***b*=the MS ID (or pseudo MS ID, or correlation) and particular communications interface from field **1100***f* (if available) of the WDR of block **2108**, and block **2142** inserts the record **2400** to queue **1980** (interface **1918**) for starting a thread **1932**. Block **2142** continues back to block **2106**. An alternate embodiment causes block **2126** to continue directly to block **2140** (no block **2138**) for a No condition from block **2126**. Regardless of whether the originating MS ID can be targeted, a correlation (in lieu of an MS ID) may be used when the MS responds with a broadcast. The WDR request made by thread **1932** can be a broadcast rather than a targeted request. Thread(s) **1932** can handle sending targeted WDR requests (to a known MS ID) and broadcast WDR requests.

Referring back to block **2118**, if it is determined the WDR does contain correlation (field **1100***m*), block **2144** peeks the CR queue **1990** (using interface **1920**) for a record **2450** containing a match (i.e. field **1100***m* matched to field **2450***b*). Thereafter, if block **2146** determines no correlation was found on queue **1990** (e.g. response took too long and entry was pruned), then processing continues to block **2120** already described. If block **2146** determines the correlation entry was found (i.e. thread **1912** received a response from an earlier request (e.g. from a thread **1922** or **1932**), then block **2148** uses date/time stamp field **2450***a* (from block **2144**) with field **1100***p* (e.g. from block **2108**) to calculate a TDOA measurement in time scale of the MS of FIG. **21** processing, and sets field **1100***f* appropriately in the WDR. Note that correlation field **2450***b* is valid across all available MS communications interfaces (e.g. all supported active wave spectrums). The TDOA measurement considers duration of time between the earlier sent date/time of record **2450** and the later time of received date/time field **1100***p*. The TDOA measurement may further be altered at block **2148** processing time to a distance knowing the velocity of the wave spectrum used as received to queue **26**. Block **2148** continues to block **2150** where the TDOA_FINAL variable is set to True, then to block **2120** for processing already described.

Referring back to block **2110**, if a WDR for a worker thread termination request was found at queue **26**, then block **2152** decrements the worker thread count by 1 (using appropriate semaphore access (e.g. **1912**-Sem)), and thread **1912** processing terminates at block **2154**. Block **2152** may also check the **1912**-Ct value, and signal the process **1912** parent thread that all worker threads are terminated when **1912**-Ct equals zero (0).

In the embodiment wherein usual MS communications data **1302** of the MS is altered to contain CK **1304** or **1314** for listening MSs in the vicinity, receive processing feeding queue **26** will place WDR information to queue **26** as CK **1304** or **1314** is detected for being present in usual communication data **1302** or **1304**. As normal communications are conducted, transmitted data **1302** or **1312** contains new data CK **1304** or **1314** to be ignored by receiving MS other character **32** processing, but to be found by listening MSs within the vicinity which anticipate presence of CK **1304** or **1314**. Otherwise, when LN-Expanse deployments have not introduced CK **1304** (or **1314**) to usual data **1302** (or **1312**) communicated on a receivable signal by MSs in the vicinity, FIG. **21** receives new data **1302** (or **1312**) sent. In any case, field **1100***p* should be as accurate as possible for when data **1302** (or **1312**) was actually received. Critical regions of code and/or anticipated execution timing may be used to affect a best setting of field **1100***p*.

So, FIG. **21** is responsible for maintaining whereabouts of others to queue **22** with data useful for triangulating itself.

FIG. **22** depicts a flowchart for describing a preferred embodiment of MS whereabouts supervisor processing, for example to ensure the MS of FIG. **22** processing (e.g. first MS) is maintaining timely whereabouts information for itself. FIG. **22** processing describes a process **1922** worker thread, and is of PIP code **6**. Thread(s) **1922** purpose is for the MS of FIG. **22** processing (e.g. a first, or sending, MS), after determining its whereabouts are stale, to periodically transmit requests for whereabouts information from MSs in the vicinity (e.g. from at least a second, or receiving, MS), and/or to start a thread **1952** for immediately determining whereabouts. Alternative embodiments to FIG. **22** will implement processing of blocks **2218** through **2224**, or processing of blocks **2226** through **2228**, or both as depicted in FIG. **22**. It is recommended that validity criteria set at block **1444** for **1922**-Max be fixed at one (1) in the preferred embodiment. Multiple channels for broadcast at block **2224** should be isolated to modular send processing feeding from a queue **24**.

In an alternative embodiment having multiple transmission channels visible to process **1922**, there can be a worker thread **1922** per channel to handle broadcasting on multiple channels. If thread(s) **1922** (block **2224**) do not transmit directly over the channel, this embodiment would provide means for communicating the channel for broadcast to send processing when interfacing to queue **24** (e.g. incorporate a channel qualifier field with WDR request inserted to queue **24**). This embodiment could allow specification of one (1) thread per channel, however multiple worker threads configurable for process **1922** as determined by the number of channels configurable for broadcast.

Processing begins at block **2202**, continues to block **2204** where the process worker thread count **1922**-Ct is accessed and incremented by 1 (using appropriate semaphore access (e.g. **1922**-Sem)), and continues to block **2206** for interim housekeeping of pruning the CR queue by invoking a Prune Queues procedure of FIG. **27**. Block **2204** may also check the **1922**-Ct value, and signal the process **1922** parent thread that all worker threads are running when **1922**-Ct reaches **1922**-Max. Block **2206** continues to block **2208** for peeking WDR queue **22** (using interface **1924**) for a special termination request entry. Thereafter, if block **2210** determines that a worker thread termination request was not found in queue **22**, processing continues to block **2212**. Block **2212** peeks the WDR queue **22** (using interface **1924**) for the most recent highest confidence entry for this MS whereabouts by searching queue **22** for: the MS ID field **1100***a* matching the MS ID of FIG. **22** processing, and a confidence field **1100***d* greater than or equal to the confidence floor value, and a most recent date/time stamp field **1100***b* within a prescribed trailing period of time of block **2212** search processing using a function of the WTV (i.e. f(WTV)=short-hand for "function of WTV") for the period. For example, block **2212** peeks the queue (i.e. makes a copy for use if an entry found for subsequent processing, but does not remove the entry from queue) for a WDR of the first MS which has the greatest confidence over 75 and has been most recently inserted to queue **22** in the last 3 seconds. Since the MS whereabouts accuracy may be dependent on timeliness of the WTV, it is recommended that the f(WTV) be some value less than or equal to WTV, but preferably not greater than the WTV. Thread **1922** is of less value to the MS when not making sure in a timely manner the MS is maintaining timely whereabouts for itself. In an alternate embodiment, a movement tolerance (e.g. user configured or system set (e.g. 3 meters)) is incorporated at the MS, or at service(s) used to locate the MS, for knowing when the MS

has significantly moved (e.g. more than 3 meters) and how long it has been (e.g. 45 seconds) since last significantly moving. In this embodiment, the MS is aware of the period of time since last significantly moving and the f(WTV) is set using the amount of time since the MS significantly moved (i.e. f(WTV)=as described above, or the amount of time since significantly moving, whichever is greater). This way a large number of (perhaps more confident candidates) WDRs are searched in the time period when the MS has not significantly moved. Optional blocks **278** through **284** may have been incorporated to FIG. **2**F for movement tolerance processing just described, in which case the LWT is compared to the current date/time to adjust the WTV for the correct trailing period. In any case, a WDR is sought at block **2212** which will verify whether or not MS whereabouts are current.

Thereafter, if block **2214** determines a satisfactory WDR was found, then processing continues to block **2216**. Block **2216** causes thread **1922** to sleep according to a f(WTV) (preferably a value less than or equal to the WTV (e.g. 95% of WTV)). When the sleep time has elapsed, processing continues back to block **2206** for another loop iteration of blocks **2206** through **2214**.

If block **2214** determines a current WDR was not found, then block **2218** builds a WDR request (e.g. containing record **2490** with field **2490***a* for the MS of FIG. **22** processing (MS ID or pseudo MS ID) so receiving MSs in the LN-expanse know who to respond to, and field **2490***b* with appropriate correlation for response), block **2220** builds a record **2450** (using correlation generated for the request at block **2218**), block **2222** inserts the record **2450** to queue **1990** (using interface **1928**), and block **2224** broadcasts the WDR request (record **2490**) for responses. Absence of field **2490***d* indicates to send processing feeding from queue **24** to broadcast on all available comm. interfaces **70**.

With reference now to FIG. **24**C, depicted is an illustration for describing a preferred embodiment of a WDR request record, as communicated to queue **24** or **26**. When a LN-expanse globally uses NTP, as found in thread **19***xx* processing described for architecture **1900**, a WDR request record **2490** may, or may not, be required. TDOA calculations can be made using a single unidirectional data (**1302** or **1312**) packet containing a sent date/time stamp (of when the data was sent) as described above.

Records **2490** contain a MS ID field **2490***a* and correlation field **2490***b*. MS ID field **2490***a* contains an MS ID (e.g. a value of field **1100***a*). An alternate embodiment will contain a pseudo MS ID (for correlation), perhaps made by a derivative of the MS ID with a unique (suffix) portion, so that receiving MSs can directly address the MS sending the request without actually knowing the MS ID (i.e. they know the pseudo MS ID which enables the MS to recognize originated transmissions). Correlation data field **2490***b* contains unique correlation data (e.g. MS id with suffix of unique number) used to provide correlation for matching sent requests (data **1302**) with received WDR responses (data **1302** or **1312**). Upon a correlation match, a TDOA measurement is calculated using the time difference between field **2450***a* and a date/time stamp of when the response was received (e.g. field **1100***p*). Received date/time stamp field **2490***c* is added by receive processing feeding queue **26** when an MS received the request from another MS. Comm interface field **2490***d* is added by receive processing inserting to queue **26** for how to respond and target the originator. Many MSs do not have choices of communications interfaces, so field **2490***d* may not be required. If available it is used, otherwise a response can be a broadcast. Field **2490***d* may contain a wave spectrum identifier for uniquely identifying how to respond (e.g. one to one with communications interface), or any other value for indicating how to send given how the request was received.

With reference back to FIG. **22**, block **2218** builds a request that receiving MSs will know is for soliciting a response with WDR information. Block **2218** generates correlation for field **2450***b* to be returned in responses to the WDR request broadcast at block **2224**. Block **2220** also sets field **2450***a* to when the request was sent. Preferably, field **2450***a* is set as close to the broadcast as possible. In an alternative embodiment, broadcast processing feeding from queue **24** makes the record **2450** and inserts it to queue **1990** with a most accurate time of when the request was actually sent. Fields **2450***a* are to be as accurate as possible. Block **2224** broadcasts the WDR request data **1302** (using send interface **1926**) by inserting to queue **24** so that send processing broadcasts data **1302**, for example as far as radius **1306**. Broadcasting preferably uses all available communications interface(s) **70** (e.g. all available wave spectrums). Therefore, the comm interface field **2490***d* is not set (which implies to send processing to do a broadcast).

Block **2224** continues to block **2226** where a record **2400** is built (i.e. field **2400***a*=**1952** and field **2400***b* is set to null (e.g. −1)) and then block **2228** inserts the record **2400** to TR queue **1980** (using interface **1930**) so that a thread **1952** will perform processing. Blocks **2226** and **2228** may be replaced with an alternative embodiment for starting a thread **1952**. Block **2228** continues back to block **2216**.

Referring back to block **2210**, if a worker thread termination request entry was found at queue **22**, then block **2230** decrements the worker thread count by 1 (using appropriate semaphore access (e.g. **1922**-Sem)), and thread **1922** processing terminates at block **2232**. Block **2230** may also check the **1922**-Ct value, and signal the process **1922** parent thread that all worker threads are terminated when **1922**-Ct equals zero (0).

In the embodiment wherein usual MS communications data **1302** of the MS is altered to contain CK **1304** for listening MSs in the vicinity, send processing feeding from queue **24**, caused by block **2224** processing, will place the request as CK **1304** embedded in usual data **1302** at the next opportune time of sending usual data **1302**. This may require the alternative embodiment of adding the entry to queue **1990** being part of send processing. As the MS conducts its normal communications, transmitted data **1302** contains new data CK **1304** to be ignored by receiving MS other character **32** processing, but to be found by listening MSs within the vicinity which anticipate presence of CK **1304**. Otherwise, when LN-Expanse deployments have not introduced CK **1304** to usual data **1302** communicated on a receivable signal by MSs in the vicinity, FIG. **22** sends new WDR request data **1302**.

FIG. **23** depicts a flowchart for describing a preferred embodiment of MS timing determination processing. FIG. **23** processing describes a process **1932** worker thread, and is of PIP code **6**. Thread(s) **1932** purpose is for the MS of FIG. **23** processing to determine TDOA measurements when needed for WDR information received. It is recommended that validity criteria set at block **1444** for **1932**-Max be set as high as possible (e.g. 12) relative performance considerations of architecture **1900**, to service multiple threads **1912**.

Processing begins at block **2302**, continues to block **2304** where the process worker thread count **1932**-Ct is accessed and incremented by 1 (using appropriate semaphore access (e.g. **1932**-Sem)), and continues to block **2306** for interim housekeeping of pruning the CR queue by invoking a Prune Queues procedure of FIG. **27**. Block **2304** may also check the **1932**-Ct value, and signal the process **1932** parent thread that all worker threads are running when **1932**-Ct reaches **1932**-Max.

Thereafter, block **2308** retrieves from queue **1980** a record **2400** (using interface **1934**), perhaps a special termination request entry, or a record **2400** received from thread(s) **1912**, and only continues to block **2310** when a record **2400** containing field **2400***a* set to **1932** has been retrieved. Block **2308** stays blocked on retrieving from queue **1980** until a record **2400** with field **2400***a*=**1932** is retrieved. If block **2310** determines a special entry indicating to terminate was not found in queue **1980**, processing continues to block **2312**.

If at block **2312**, the record **2400** does not contain a MS ID (or pseudo MS ID) in field **2400***b*, processing continues to block **2314** for building a WDR request (record **2490**) to be broadcast, and then to block **2318**. Broadcasting preferably uses all available communications interface(s) **70** (e.g. all available wave spectrums). If block **2312** determines the field **2400***b* is a valid MS ID (not null), block **2316** builds a WDR request targeted for the MS ID, and processing continues to block **2318**. A targeted request is built for targeting the MS ID (and communications interface, if available) from field **2400***b*. Send processing is told which communications interface to use, if available (e.g. MS has multiple), otherwise send processing will target each available interface. In the unlikely case a MS ID is present in field **2400***b* without the communications interface applicable, then all communications interfaces **70** are used with the targeted MS ID. In MS embodiments with multiple communications interfaces **70**, then **2400***b* is to contain the applicable communication interface for sending. Block **2318** generates appropriate correlation for a field **2450***b* (e.g. to be compared with a response WDR at block **2144**), block **2320** sets field **2450***a* to the current MS date/time stamp, block **2322** inserts the record **2450** to queue **1990** (using interface **1936**), and block **2324** sends/broadcasts (using interface **1938**) a WDR request (record **2490**). Thereafter, processing continues back to block **2306** for another loop iteration. An alternative embodiment will only target a WDR request to a known MS ID. For example, block **2312** would continue back to block **2306** if no MS ID is found (=null), otherwise it will continue to block **2316** (i.e. no use for block **2314**).

Block **2318** sets field **2450***b* to correlation to be returned in responses to the WDR request sent/broadcast at block **2324**. Block **2320** sets field **2450***a* to when the request is sent. Preferably, field **2450***a* is set as close as possible to when a send occurred. In an alternative embodiment, send processing feeding from queue **24** makes the record **2450** and inserts it to queue **1990** with a most accurate time of when the request was actually sent. Fields **2450***a* are to be as accurate as possible. Block **2324** sends/broadcasts the WDR request data **1302** (using send interface **1938**) by inserting to queue **24** a record **2490** (**2490***a*=the targeted MS ID (or pseudo MS ID) OR null if arrived to from block **2314**, field **2490***b*=correlation generated at block **2318**) so that send processing sends data **1302**, for example as far as radius **1306**. A null MS ID may be responded to by all MSs in the vicinity. A non-null MS ID is to be responded to by a particular MS. Presence of field **2490***d* indicates to send processing feeding from queue **24** to target the MS ID over the specified comm. interface (e.g. when MS has a plurality of comm. interfaces **70** (e.g. cellular, WiFi, Bluetooth, etc; i.e. MS supports multiple classes of wave spectrum)).

Referring back to block **2310**, if a worker thread termination request was found at queue **1980**, then block **2326** decrements the worker thread count by 1 (using appropriate semaphore access (e.g. **1932**-Sem)), and thread **1932** processing terminates at block **2328**. Block **2326** may also check the **1932**-Ct value, and signal the process **1932** parent thread that all worker threads are terminated when **1932**-Ct equals zero (0).

In the embodiment wherein usual MS communications data **1302** of the MS is altered to contain CK **1304** for listening MSs in the vicinity, send processing feeding from queue **24**, caused by block **2324** processing, will place the WDR request as CK **1304** embedded in usual data **1302** at the next opportune time of sending usual data **1302**. As the MS conducts its normal communications, transmitted data **1302** contains new data CK **1304** to be ignored by receiving MS other character **32** processing, but to be found by listening MSs within the vicinity which anticipate presence of CK **1304**. This may require the alternative embodiment of adding the entry to queue **1990** being part of send processing. Otherwise, when LN-Expanse deployments have not introduced CK **1304** to usual data **1302** communicated on a receivable signal by MSs in the vicinity, FIG. **22** sends/broadcasts new WDR request data **1302**.

An alternate embodiment to block **2324** can wait for a response with a reasonable timeout, thereby eliminating the need for blocks **2318** through **2322** which is used to correlate the subsequent response (to thread **1912**) with the request sent at block **2324**. However, this will cause a potentially unpredictable number of simultaneously executing thread(s) **1932** when many MSs are in the vicinity.

Thread(s) **1932** are useful when one or both parties to WDR transmission (sending and receiving MS) do not have NTP enabled. TDOA measurements are taken to triangulate the MS relative other MSs in real time.

FIG. **25** depicts a flowchart for describing a preferred embodiment of MS WDR request processing, for example when a remote MS requests (e.g. from FIG. **22** or **23**) a WDR. Receive processing identifies targeted requests destined (e.g. FIG. **23**) for the MS of FIG. **25** processing, and identifies general broadcasts (e.g. FIG. **22**) for processing as well. FIG. **25** processing describes a process **1942** worker thread, and is of PIP code **6**. Thread(s) **1942** purpose is for the MS of FIG. **25** processing to respond to incoming WDR requests. It is recommended that validity criteria set at block **1444** for **1942**-Max be set as high as possible (e.g. 10) relative performance considerations of architecture **1900**, to service multiple WDR requests simultaneously. Multiple channels for receiving data fed to queue **26** should be isolated to modular receive processing.

In an alternative embodiment having multiple receiving transmission channels visible to process **1942**, there can be a worker thread **1942** per channel to handle receiving on multiple channels simultaneously. If thread(s) **1942** do not receive directly from the channel, the preferred embodiment of FIG. **25** would not need to convey channel information to thread(s) **1942** waiting on queue **24** anyway. Embodiments could allow specification/configuration of many thread(s) **1942** per channel.

Processing begins at block **2502**, continues to block **2504** where the process worker thread count **1942**-Ct is accessed and incremented by 1 (using appropriate semaphore access (e.g. **1942**-Sem)), and continues to block **2506** for retrieving from queue **26** a record **2490** (using interface **1948**), perhaps a special termination request entry, and only continues to block **2508** when a record **2490** is retrieved. Block **2506** stays blocked on retrieving from queue **26** until any record **2490** is retrieved. If block **2508** determines a special entry indicating to terminate was not found in queue **26**, processing continues to block **2510**. There are various embodiments for thread(s) **1912** and thread(s) **1942** to feed off a queue **26** for different record types, for example, separate queues **26**A and **26**B, or a

thread target field with either record found at queue **26** (e.g. like field **2400***a*). In another embodiment, thread(s) **1912** are modified with logic of thread(s) **1942** to handle all records described for a queue **26**, since thread(s) **1912** are listening for queue **26** data anyway.

Block **2510** peeks the WDR queue **22** (using interface **1944**) for the most recent highest confidence entry for this MS whereabouts by searching queue **22** for: the MS ID field **1100***a* matching the MS ID of FIG. **25** processing, and a confidence field **1100***d* greater than or equal to the confidence floor value, and a most recent date/time stamp field **1100***b* within a prescribed trailing period of time of block **2510** search processing (e.g. 2 seconds). For example, block **2510** peeks the queue (i.e. makes a copy for use if an entry found for subsequent processing, but does not remove the entry from queue) for a WDR of the MS (of FIG. **25** processing) which has the greatest confidence over 75 and has been most recently inserted to queue **22** in the last 2 seconds. It is recommended that the trailing period of time used by block **2510** be never greater than a few seconds. Thread **1942** is of less value to the LN-expanse when it responds with outdated/invalid whereabouts of the MS to facilitate locating other MSs. In an alternate embodiment, a movement tolerance (e.g. user configured or system set (e.g. 3 meters)) is incorporated at the MS, or at service(s) used to locate the MS, for knowing when the MS has significantly moved (e.g. more than 3 meters) and how long it has been (e.g. 45 seconds) since last significantly moving. In this embodiment, the MS is aware of the period of time since last significantly moving and the trailing period of time used by block **2510** is set using the amount of time since the MS significantly moved, or the amount of time since significantly moving, whichever is greater. This way a large number of (perhaps more confident candidate) WDRs are searched in the time period when the MS has not significantly moved. Optional blocks **278** through **284** may have been incorporated to FIG. **2**F for movement tolerance processing just described, in which case the LWT is compared to the current date/time to adjust the trailing period of time used by block **2510** for the correct trailing period. In any case, a WDR is sought at block **2510** to satisfy a request helping another MS in the LN-expanse locate itself.

Thereafter, if block **2512** determines a useful WDR was not found, then processing continues back to block **2506** for another loop iteration of processing an inbound WDR request. If block **2512** determines a useful WDR was found, then block **2514** prepares the WDR for send processing with correlation field **1100***m* set from correlation field **2490***b* retrieved at block **2506**, and block **2516** sends/broadcasts (per field **2490***a*) the WDR information (using send interface **1946**) by inserting to queue **24** so that send processing transmits data **1302**, for example as far as radius **1306**, and processing continues back to block **2506**. At least fields **1100***b*, **1100***c*, **1100***d*, **1100***m* and **1100***n* are sent/broadcast. See FIG. **11**A descriptions. Fields are set to the following upon exit from block **2514**:

MS ID field **1100***a* is preferably set with: Field **2490***a* from queue **26**.

DATE/TIME STAMP field **1100***b* is preferably set with: Field **1100***b* from queue **22**.

LOCATION field **1100***c* is preferably set with: Field **1100***c* from queue **22**.

CONFIDENCE field **1100***d* is preferably set with: Field **1100***d* from queue **22**.

LOCATION TECHNOLOGY field **1100***e* is preferably set with: Field **1100***e* from queue **22**.

LOCATION REFERENCE INFO field **1100***f* is preferably set with: null (not set) for Broadcast by send processing, otherwise set to field **2490***d* for Send by send processing.

COMMUNICATIONS REFERENCE INFO field **1100***g* is preferably set with: null (not set).

SPEED field **1100***h* is preferably set with: Field **1100***h* from queue **22**.

HEADING field **1100***i* is preferably set with: Field **1100***i* from queue **22**.

ELEVATION field **1100***j* is preferably set with: Field **1100***j* from queue **22**.

APPLICATION FIELDS field **1100***k* is preferably set with: Field **1100***k* from queue **22**. An alternate embodiment will add, alter, or discard data (with or without date/time stamps) here at the time of block **2514** processing.

CORRELATION FIELD **1100***m* is preferably set with: Field **2490***b* from queue **26**.

SENT DATE/TIME STAMP field **1100***n* is preferably set with: Sent date/time stamp as close in processing the send/broadcast of block **2516** as possible.

RECEIVED DATE/TIME STAMP field **1100***p* is preferably set with: Not Applicable (i.e. N/A for sending).

Embodiments may rely completely on the correlation field **2490***b* with no need for field **2490***a*. Referring back to block **2508**, if a worker thread termination request was found at queue **26**, then block **2518** decrements the worker thread count by 1 (using appropriate semaphore access (e.g. **1942**-Sem)), and thread **1942** processing terminates at block **2520**. Block **2518** may also check the **1942**-Ct value, and signal the process **1942** parent thread that all worker threads are terminated when **1942**-Ct equals zero (0).

Block **2516** causes sending/broadcasting data **1302** containing CK **1304**, depending on the type of MS, wherein CK **1304** contains WDR information prepared as described above for block **2514**. Alternative embodiments of block **2510** may not search a specified confidence value, and broadcast the best entry available anyway so that listeners in the vicinity will decide what to do with it. A semaphore protected data access (instead of a queue peek) may be used in embodiments where there is always one WDR current entry maintained for the MS.

In the embodiment wherein usual MS communications data **1302** of the MS is altered to contain CK **1304** for listening MSs in the vicinity, send processing feeding from queue **24**, caused by block **2516** processing, will place WDR information as CK **1304** embedded in usual data **1302** at the next opportune time of sending usual data **1302**. If an opportune time is not timely, send processing should discard the send request of block **2516** to avoid broadcasting outdated whereabouts information (unless using a movement tolerance and time since last significant movement). As the MS conducts its normal communications, transmitted data **1302** contains new data CK **1304** to be ignored by receiving MS other character **32** processing, but to be found by listening MSs within the vicinity which anticipate presence of CK **1304**. Otherwise, when LN-Expanse deployments have not introduced CK **1304** to usual data **1302** communicated on a receivable signal by MSs in the vicinity, FIG. **25** sends/broadcasts new WDR response data **1302**. In any case, field **1100***n* should be as accurate as possible for when data **1302** is actually sent. Critical regions of code (i.e. prevent thread preemption) and/or anticipated execution timing may be used to affect a best setting of field **1100***n*.

In an alternate embodiment, records **2490** contain a sent date/time stamp field **2490***e* of when the request was sent by a remote MS, and the received date/time stamp field **2490***c* is processed at the MS in FIG. **25** processing. This would enable

block **2514** to calculate a TDOA measurement for returning in field **1100***f* of the WDR sent/broadcast at block **2516**.

FIG. **26**A depicts a flowchart for describing a preferred embodiment of MS whereabouts determination processing. FIG. **26**A processing describes a process **1952** worker thread, and is of PIP code **6**. Thread(s) **1952** purpose is for the MS of FIG. **26**A processing to determine its own whereabouts with useful WDRs from other MSs. It is recommended that validity criteria set at block **1444** for **1952**-Max be set as high as possible (e.g. 10) relative performance considerations of architecture **1900**, to service multiple threads **1912**. **1952**-Max may also be set depending on what DLM capability exists for the MS of FIG. **26**A processing. In an alternate embodiment, thread(s) **19***xx* are automatically throttled up or down (e.g. **1952**-Max) per unique requirements of the MS as it travels.

Processing begins at block **2602**, continues to block **2604** where the process worker thread count **1952**-Ct is accessed and incremented by 1 (using appropriate semaphore access (e.g. **1952**-Sem)), and continues to block **2606** for interim housekeeping of pruning the WDR queue by invoking a Prune Queues procedure of FIG. **27**. Block **2604** may also check the **1952**-Ct value, and signal the process **1952** parent thread that all worker threads are running when **1952**-Ct reaches **1952**-Max. Block **2606** may not be necessary since pruning may be accomplished at block **2620** when invoking FIG. **2**F (block **292**).

Thereafter, block **2608** retrieves from queue **1980** a record **2400** (using interface **1958**), perhaps a special termination request entry, or a record **2400** received from thread(s) **1912**, and only continues to block **2610** when a record **2400** containing field **2400***a* set to **1952** has been retrieved. Block **2608** stays blocked on retrieving from queue **1980** until a record **2400** with field **2400***a*=**1952** is retrieved. If block **2610** determines a special entry indicating to terminate was not found in queue **1980**, processing continues to block **2612**.

Block **2612** peeks the WDR queue **22** (using interface **1954**) for the most recent highest confidence entry for this MS whereabouts by searching queue **22** for: the MS ID field **1100***a* matching the MS ID of FIG. **26**A processing, and a confidence field **1100***d* greater than or equal to the confidence floor value, and a most recent date/time stamp field **1100***b* within a prescribed trailing period of time of block **2612** search processing using a f(WTV) for the period. For example, block **2612** peeks the queue (i.e. makes a copy for use if an entry found for subsequent processing, but does not remove the entry from queue) for a WDR of the MS (of FIG. **26**A processing) which has the greatest confidence over 75 and has been most recently inserted to queue **22** in the last 2 seconds. Since MS whereabouts accuracy may be dependent on timeliness of the WTV, it is recommended that the f(WTV) be some value less than or equal to WTV. In an alternate embodiment, a movement tolerance (e.g. user configured or system set (e.g. 3 meters)) is incorporated at the MS, or at service(s) used to locate the MS, for knowing when the MS has significantly moved (e.g. more than 3 meters) and how long it has been (e.g. 45 seconds) since last significantly moving. In this embodiment, the MS is aware of the period of time since last significantly moving and the f(WTV) is set using the amount of time since the MS significantly moved (i.e. f(WTV)=as described above, or the amount of time since significantly moving, whichever is greater). This way a large number of (perhaps more confident candidate) WDRs are searched in the time period when the MS has not significantly moved. Optional blocks **278** through **284** may have been incorporated to FIG. **2**F for movement tolerance processing just described, in which case the LWT is compared to the current date/time to adjust the WTV for the correct trailing period.

Thereafter, if block **2614** determines a timely whereabouts for this MS already exists to queue **22** (current WDR found), then processing continues back to block **2606** for another loop iteration of processing. If **2614** determines a satisfactory WDR does not already exist in queue **22**, then block **2600** determines a new highest confidence WDR for this MS (FIG. **26**B processing) using queue **22**.

Thereafter, if block **2616** determines a WDR was not created (BESTWDR variable=null) for the MS of FIG. **26**A processing (by block **2600**), then processing continues back to block **2606**. If block **2616** determines a WDR was created (BESTWDR=WDR created by FIG. **26**B) for the MS of FIG. **26**A processing by block **2600**, then processing continues to block **2618** for preparing FIG. **2**F parameters and FIG. **2**F processing is invoked with the new WDR at block **2620** (for interface **1956**) before continuing back to block **2606**. Parameters set at block **2618** are: WDRREF=a reference or pointer to the WDR completed at block **2600**; DELETEQ=FIG. **26**A location queue discard processing; and SUPER=FIG. **26**A supervisory notification processing.

Referring back to block **2610**, if a worker thread termination request was found at queue **1980**, then block **2622** decrements the worker thread count by 1 (using appropriate semaphore access (e.g. **1952**-Sem)), and thread **1952** processing terminates at block **2624**. Block **2622** may also check the **1952**-Ct value, and signal the process **1952** parent thread that all worker threads are terminated when **1952**-Ct equals zero (0).

Alternate embodiments to FIG. **26**A will have a pool of thread(s) **1952** per location technology (WDR field **1100***e*) for specific WDR field(s) selective processing. FIG. **26**A processing is shown to be generic with handling all WDRs at block **2600**.

FIG. **26**B depicts a flowchart for describing a preferred embodiment of processing for determining a highest possible confidence whereabouts, for example in ILM processing, such as processing of FIG. **26**A block **2600**. Processing starts at block **2630**, and continues to block **2632** where variables are initialized (BESTWDR=null, THIS_MS=null, REMOTE_MS=null). BESTWDR will reference the highest confidence WDR for whereabouts of the MS of FIG. **26**B processing (i.e. this MS) upon return to FIG. **26**A when whereabouts determination is successful, otherwise BESTWDR is set to null (none found). THIS_MS points to an appropriately sorted list of WDRs which were originated by this MS and are DLM originated (i.e. inserted by the DLM of FIG. **26**B processing). REMOTE_MS points to an appropriately sorted list of WDRs which were originated by other MSs (i.e. from DLMs and/or ILMs and collected by the ILM of FIG. **26**B processing).

Thereafter, block **2634** peeks the WDR queue **22** (using interface **1954**) for most recent WDRs by searching queue **22** for: confidence field **1100***d* greater than or equal to the confidence floor value, and a most recent date/time stamp field **1100***b* within a prescribed trailing period of time of block **2634** search processing using a f(WTV) for the period. For example, block **2634** peeks the queue (i.e. makes a copy of all WDRs to a result list for use if any found for subsequent processing, but does not remove the entry(s) from queue) for all WDRs which have confidence over 75 and has been most recently inserted to queue **22** in the last 2 seconds. It is recommended that the f(WTV) used here be some value less than or equal to the WTV (want to be ahead of curve, so may use a percentage (e.g. 90%)), but preferably not greater than

a couple/few seconds (depends on MS, MS applications, MS environment, whereabouts determination related variables, etc).

In an alternative embodiment, thread(s) **1952** coordinate with each other to know successes, failures or progress of their sister threads for automatically adjusting the trailing f(WTV) period of time appropriately. See "Alternative IPC Embodiments" below.

Thread **1952** is of less value to the MS when whereabouts are calculated using stale WDRs, or when not enough useful WDRs are considered. In an alternate embodiment, a movement tolerance (e.g. user configured or system set (e.g. 3 meters)) is incorporated at the MS, or at service(s) used to locate the MS, for knowing when the MS has significantly moved (e.g. more than 3 meters) and how long it has been (e.g. 45 seconds) since last significantly moving. In this embodiment, the MS is aware of the period of time since last significantly moving and the f(WTV) is set using the amount of time since the MS significantly moved (i.e. f(WTV)=as described above, or the amount of time since significantly moving, whichever is greater). This way a large number of (perhaps more confident candidates) WDRs are searched in the time period when the MS has not significantly moved. Optional blocks **278** through **284** may have been incorporated to FIG. **2**F for movement tolerance processing just described, in which case the LWT is compared to the current date/time to adjust the WTV for the correct trailing period. In any case, all useful WDRs are sought at block **2634** and placed into a list upon exit from block **2634**.

Thereafter, block **2636** sets THIS_MS list and REMOTE_MS list sort keys to be used at blocks **2644** and **2654**. Blocks **2638** through **2654** will prioritize WDRs found at block **2634** depending on the sort keys made at block **2636**. A number of variables may be used to determine the best sort keys, such as the time period used to peek at block **2634** and/or the number of entries in the WDR list returned by block **2634**, and/or other variables. When the time period of search is small (e.g. less than a couple seconds), lists (THIS_MS and REMOTE_MS) should be prioritized primarily by confidence (fields **1100***d*) since any WDRs are valuable for determining whereabouts. This is the preferred embodiment.

When the time period is great, careful measure must be taken to ensure stale WDRs are not used (e.g. > few seconds, and not considering movement tolerance). Depending on decision embodiments, there will be preferred priority order sort keys created at exit from block **2636**, for example "key**1**/key**2**/key**3**" implies that "key**1**" is a primary key, "key**2**" is a second order key, and "key**3**" is a third order key. A key such as "field-**1100***b*/field-**1100***d*/field-**1100***f*:signal-strength" would sort WDRs first by using date/time stamp fields **1100***b*, then by confidence value fields **1100***d* (sorted within matching date/time stamp WDRs), then by signal-strength field **1100***f* sub-field values (sorted within matching WDR confidences; no signal strength present=lowest priority). Another sort key may be "field-**1100***d*/field-**1100***b*" for sorting WDRs first by using confidence values, then by date/time stamps (sorted within matching WDR confidences). The same or different sort keys can be used for lists THIS_MS and REMOTE_MS. Any WDR data (fields or subfields) can be sorted with a key, and sort keys can be of N order dimension such that "key**1**/key**2**/ . . . /keyN". Whatever sort keys are used, block **2686** will have to consider confidence versus being stale, relative to the WTV. In the preferred embodiment, the REMOTE_MS and THIS_MS lists are set with the same sort keys of "field-**1100***d*/field-**1100***b*" (i.e. peek time period used at block **2634** is less than 2 seconds) so that confidence is primary.

Thereafter, block **2638** gets the first (if any) WDR in the list returned at block **2634** (also processes next WDR in list when encountered again in loop of blocks **2638** through **2654**), and block **2640** checks if all WDRs have already been processed. If block **2640** finds that all WDRs have not been processed, then block **2642** checks the WDR origination. If block **2642** determines the WDR is one that originated from a remote MS (i.e. MS ID does not match the MS of FIG. **26**B processing), then block **2644** inserts the WDR into the REMOTE_MS list using the desired sort key (confidence primary, time secondary) from block **2636**, and processing continues to block **2638** for another loop iteration. If block **2642** determines the WDR is one that originated from this MS (MS ID field **1100***a* matches the MS of FIG. **26**B processing (e.g. this MS being a DLM at the time of WDR creation (this MS ID=field **1100***a*) or this MS being an ILM at the time of WDR creation (previous processing of FIG. **26**A)), then processing continues to block **2646** to determine how to process the WDR which was inserted by "this MS" for its own whereabouts.

Block **2646** accesses field **1100***f* for data found there (e.g. FIGS. **2**D and **2**E may have inserted useful TDOA measurements, even though DLM processing occurred; or FIG. **3**C may have inserted useful TDOA and/or AOA measurements with reference station(s) whereabouts; or receive processing may have inserted AOA and related measurements). Thereafter, if block **2648** determines presence of TDOA and/or AOA data, block **2650** checks if reference whereabouts (e.g. FIG. **3**C selected stationary reference location(s)) is also stored in field **1100***f*. If block **2650** determines whereabouts information is also stored to field **1100***f*, then block **2652** makes new WDR(s) from the whereabouts information containing at least the WDR Core and field **1100***f* containing the AOA and/or TDOA information as though it were from a remote DLM or ILM. Block **2652** also performs the expected result of inserting the WDR of loop processing into the THIS_MS list using the desired sort key from block **2636**. Processing then continues to block **2644** where the newly made WDR(s) is inserted into the REMOTE_MS list using the desired sort key (confidence primary, time secondary) from block **2636**. Block **2644** continues back to block **2638**.

Block **2646** through **2652** show that DLM stationary references may contribute to determining whereabouts of the MS of FIG. **26**B processing by making such references appear to processing like remote MSs with known whereabouts. Any DLM location technology processing discussed above can facilitate FIG. **26**B whereabouts processing when reference whereabouts can be maintained to field **1100***f* along with relative AOA, TDOA, MPT, confidence, and/or other useful information for locating the MS. Various embodiments will populate field **1100***f* wherever possible with any useful locating fields (see data discussed for field **1100***f* with FIG. **11**A discussions above) for carrying plenty of information to facilitate FIG. **26**B processing.

Referring back to block **2650**, if it is determined that whereabouts information was not present with the AOA and/or TDOA information of field **1100***f*, then processing continues to block **2644** for inserting into the REMOTE_MS list (appropriately with sort key from block **2636**) the currently looped WDR from block **2634**. In-range location technology associates the MS with the antenna (or cell tower) location, so that field **1100***c* already contains the antenna (or cell tower) whereabouts, and the TDOA information was stored to determine how close the MS was to the antenna (or cell tower) at the time. The WDR will be more useful in the REMOTE_MS

list, then if added to the THIS_MS list (see loop of blocks **2660** through **2680**). Referring back to block **2648**, if it is determined that no AOA and/or TDOA information was in field **1100***f*, then processing continues to block **2654** for inserting the WDR into the THIS_MS list (appropriately with sort key (confidence primary, time secondary) from block **2636**).

Block **2654** handles WDRs that originated from the MS of FIG. **26**B (this MS), such as described in FIGS. **2**A through **9**B, or results from previous FIG. **26**A processing. Block **2644** maintains remote DLMs and/or ILMs (their whereabouts) to the REMOTE_MS list in hope WDRs contain useful field **1100***f* information for determining the whereabouts of the MS of FIG. **26**B processing. Block **2652** handles WDRs that originated from the MS of FIG. **26**B processing (this MS), but also processes fields from stationary references used (e.g. FIG. **3**C) by this MS which can be helpful as though the WDR was originated by a remote ILM or DLM. Thus, block **2652** causes inserting to both lists (THIS_MS and REMOTE_MS) when the WDR contains useful information for both. Blocks **2652**, **2654** and **2644** cause the iterative loop of blocks **2660** through **2680** to perform ADLT using DLMs and/or ILMs. Alternate embodiments of blocks **2638** through **2654** may use peek methodologies to sort from queue **22** for the REMOTE_MS and THIS_MS lists.

Referring back to block **2640**, if it is determined that all WDRs in the list from block **2634** have been processed, then block **2656** initializes a DISTANCE list and ANGLE list each to null, block **2658** sets a loop iteration pointer to the first entry of the prioritized REMOTE_MS list (e.g. first entry higher priority then last entry in accordance with sort key used), and block **2660** starts the loop for working with ordered WDRs of the REMOTE_MS list. Exit from block **2640** to block **2656** occurs when the REMOTE_MS and THIS_MS lists are in the desired priority order for subsequent processing. Block **2660** gets the next (or first) REMOTE_MS list entry for processing before continuing to block **2662**. If block **2662** determines all WDRs have not yet been processed from the REMOTE_MS list, then processing continues to block **2664**.

Blocks **2664** and **2670** direct collection of all useful ILM triangulation measurements for TDOA, AOA, and/or MPT triangulation of this MS relative known whereabouts (e.g. other MSs). It is interesting to note that TDOA and AOA measurements (field **1100***f*) may have been made from different communications interfaces **70** (e.g. different wave spectrums), depending on interfaces the MS has available (i.e. all can participate). For example, a MS with blue-tooth, WiFi and cellular phone connectivity (different class wave spectrums supported) can be triangulated using the best available information (i.e. heterogeneous location technique). Examination of fields **1100***f* in FIG. **17** can show wave spectrums (and/or particular communications interfaces **70**) inserted by receive processing for what the MS supports. If block **2664** determines an AOA measurement is present (field **1100***f* sub-field), then block **2666** appends the WDR to the ANGLE list, and processing continues to block **2668**. If block **2664** determines an AOA measurement is not present, then processing continues to block **2670**. If block **2670** determines a TDOA measurement is present (field **1100***f* sub-field), then block **2672** appends the WDR to the DISTANCE list, and processing continues to block **2674**. Block **2674** uses WDRs for providing at least an in-range whereabouts of this MS by inserting to the THIS_MS list in sorted confidence priority order (e.g. highest confidence first in list, lowest confidence at end of list). Block **2674** continues to block **2668**. Block **2674** may cause duplicate WDR(s) inserted to the THIS_MS list, but this will have no negative effect on selected outcome.

Block **2668** compares the ANGLE and DISTANCE lists constructed thus far from loop processing (blocks **2660** through **2680**) with minimum triangulation requirements (e.g. see "Missing Part Triangulation (MPT)" above). Three (3) sides, three (3) angles and a side, and other known triangular solution guides will also be compared. Thereafter, if block **2676** determines there is still not enough data to triangulate whereabouts of this MS, then processing continues back to block **2660** for the next REMOTE_MS list entry, otherwise block **2678** maximizes diversity of WDRs to use for triangulating. Thereafter, block **2680** uses the diversified DISTANCE and ANGLE lists to perform triangulation of this MS, block **2682** inserts the newly determined WDR into the THIS_MS list in sort key order, and continues back to block **2660**. Block **2680** will use heterogeneous (MPT), TDOA and/or AOA triangulation on ANGLE and DISTANCE lists for determining whereabouts.

Block **2682** preferably keeps track of (or checks THIS_MS for) what it has thus far determined whereabouts for in this FIG. **26**B thread processing to prevent inserting the same WDR to THIS_MS using the same REMOTE_MS data. Repeated iterations of blocks **2676** through **2682** will see the same data from previous iterations and will use the best of breed data in conjunction with each other at each iteration (in current thread context). While inserting duplicates to THIS_MS at block **2682** does not cause failure, it may be avoided for performance reasons. Duplicate insertions are preferably avoided at block **2674** for performance reasons as well, but they are again not harmful. Block **2678** preferably keeps track of previous diversity order in this FIG. **26**B thread processing to promote using new ANGLE and DISTANCE data in whereabouts determination at block **2680** (since each iteration is a superset of a previous iteration (in current thread context). Block **2678** promotes using WDRs from different MSs (different MS IDs), and from MSs located at significantly different whereabouts (e.g. to maximize surroundedness), preferably around the MS of FIG. **26**B processing. Block **2678** preferably uses sorted diversity pointer lists so as to not affect actual ANGLE and DISTANCE list order. The sorted pointer lists provide pointers to entries in the ANGLE and DISTANCE lists for a unique sorted order governing optimal processing at block **2680** to maximize unique MSs and surrounded-ness, without affecting the lists themselves (like a SQL database index). Different embodiments of blocks **2678** through **2682** should minimize inserting duplicate WDRs (for performance reasons) to THIS_MS which were determined using identical REMOTE_MS list data. Block **2682** causes using ADLT at blocks **2684** through **2688** which uses the best of breed whereabouts, either as originated by this MS maintained in THIS_MS list up to the thread processing point of block **2686**, or as originated by remote MSs (DLMs and/or ILMs) processed by blocks **2656** through the start of block **2684**.

Referring back to block **2662**, if it is determined that all WDRs in the REMOTE_MS list have been processed, then block **2684** sets the BESTWDR reference to the head of THIS_MS (i.e. BESTWDR references first WDR in THIS_MS list which is so far the best candidate WDR (highest confidence) for this MS whereabouts, or null if the list is empty). It is possible that there are other WDRs with matching confidence adjacent to the highest confidence entry in the THIS_MS list. Block **2684** continues to block **2686** for comparing matching confidence WDRs, and if there are matches, then breaking a tie between WDRs with matching confidence by consulting any other WDR field(s) (e.g. field **1100***f* signal

strength, or location technology field **1100***e*, etc). If there is still a tie between a plurality of WDRs, then block **2686** may average whereabouts to the BESTWDR WDR using the matching WDRs. Thereafter processing continues to block **2688** where the BESTWDR is completed, and processing terminates at block **2690**. Block **2688** also frees resources (if any) allocated by FIG. **26**B processing (e.g. lists). Blocks **2686** through **2688** result in setting BESTWDR to the highest priority WDR (i.e. the best possible whereabouts determined). It is possible that FIG. **26**B processing causes a duplicate WDR inserted to queue **22** (at block **2620**) for this MS whereabouts determination, but that is no issue except for impacting performance to queue **22**. An alternate embodiment to queue **22** may define a unique index for erring out when inserting a duplicate to prevent frivolous duplicate entries, or block **2688** will incorporate processing to eliminate the chance of inserting a WDR of less use than what is already contained at queue **22**. Therefore, block **2688** may include processing for ensuring a duplicate will not be inserted (e.g. null the BESTWDR reference) prior to returning to FIG. **26**A at block **2690**.

Averaging whereabouts at block **2686** occurs only when there are WDRs at the head of the list with a matching highest confidence value and still tie in other WDR fields consulted, yet whereabouts information is different. In this case, all matching highest confidence whereabouts are averaged to the BESTWDR to come up with whereabouts in light of all matching WDRs. Block **2686** performs ADLT when finalizing a single whereabouts (WDR) using any of the whereabouts found in THIS_MS (which may contain at this point DLM whereabouts originated by this MS and/or whereabouts originated by remote DLMs and/or ILMs). Block **2686** must be cognizant of sort keys used at blocks **2652** and **2654** in case confidence is not the primary key (time may be primary).

If no WDRs were found at block **2634**, or no THIS_MS list WDRs were found at blocks **2652** and **2654**, and no REMOTE_MS list entries were found at block **2644**; or no THIS_MS list WDRs were found at blocks **2652** and **2654**, and no REMOTE_MS list entries were found useful at blocks **2664** and/or **2670**; then block **2684** may be setting BESTWDR to a null reference (i.e. none in list) in which case block **2686** does nothing. Hopefully, at least one good WDR is determined for MS whereabouts and a new WDR is inserted for this MS to queue **22**, otherwise a null BESTWDR reference will be returned (checked at block **2616**). See FIG. **11**A descriptions. If BESTWDR is not null, then fields are set to the following upon exit from block **2688**:

MS ID field **1100***a* is preferably set with: MS ID of MS of FIG. **26**B processing.

DATE/TIME STAMP field **1100***b* is preferably set with: Date/time stamp of block **2688** processing.

LOCATION field **1100***c* is preferably set with: Resulting whereabouts after block **2688** completion.

CONFIDENCE field **1100***d* is preferably set with: WDR Confidence at THIS_MS list head.

LOCATION TECHNOLOGY field **1100***e* is preferably set with: "ILM TDOA Triangulation", "ILM AOA Triangulation", "ILM MPT Triangulation" or "ILM in-range", as determined by the WDRs inserted to MS_LIST at blocks **2674** and **2682**. The originator indicator is set to ILM.

LOCATION REFERENCE INFO field **1100***f* is preferably set with: null (not set), but may be set with contributing data for analysis of queue **22** provided it is marked for being overlooked by future processing of blocks **2646** and **2648** (e.g. for debug purpose).

COMMUNICATIONS REFERENCE INFO field **1100***g* is preferably set with: null (not set).

SPEED field **1100***h* is preferably set with: Block **2688** may compare prioritized entries and their order of time (field **1100***b*) in THIS_MS list for properly setting this field, if possible.

HEADING field **1100***i* is preferably set with: null (not set). Block **2688** may compare prioritized entries and their order of time (field **1100***b*) in THIS_MS list for properly setting this field, if possible.

ELEVATION field **1100***j* is preferably set with: Field **1100***j* of BESTWDR (may be averaged if WDR tie(s)), if available.

APPLICATION FIELDS field **1100***k* is preferably set with: Field(s) **1100***k* from BESTWDR or tie(s) thereof from THIS_MS. An alternate embodiment will add, alter, or discard data (with or without date/time stamps) here at the time of block **2688** processing.

CORRELATION FIELD **1100***m* is preferably set with: Not Applicable (i.e. not maintained to queue **22**).

SENT DATE/TIME STAMP field **1100***n* is preferably set with: Not Applicable (i.e. not maintained to queue **22**).

RECEIVED DATE/TIME STAMP field **1100***p* is preferably set with: Not Applicable (i.e. not maintained to queue **22**).

Block **2680** determines whereabouts using preferred guidelines, such as whereabouts determined never results in a confidence value exceeding any confidence value used to determine whereabouts. Some embodiments will use the mean (average) of confidence values used, some will use the highest, and some the lowest of the WDRs used. Preferred embodiments tend to properly skew confidence values to lower values as the LN-Expanse grows away from region **1022**. Blocks **2668** through **2680** may consult any of the WDR fields (e.g. field **1100***f* sub-fields yaw, pitch, roll; speed, heading, etc) to deduce the most useful WDR inputs for determining an optimal WDR for this MS whereabouts.

Alternative IPC Embodiments

Thread(s) **1952** are started for every WDR collected from remote MSs. Therefore, it is possible that identical new WDRs are inserted to queue **22** using the same WDR information at blocks **2634** of simultaneously executing threads **1952**, but this will not cause a problem since at least one will be found when needed, and duplicates will be pruned together when appropriate. Alternative embodiments provide IPC (Interprocess Communications Processing) coordination between **1952** threads for higher performance processing, for example:

- As mentioned above, thread(s) **1952** can coordinate with each other to know successes, failures or progress of their sister **1952** thread(s) for automatically adjusting the trailing f(WTV) period of time appropriately. The f(WTV) period of time used at block **2634** would be semaphore accessed and modified (e.g. increased) for another **1952** thread when a previous **1952** thread was unsuccessful in determining whereabouts (via semaphore accessed thread outcome indicator). After a successful determination, the f(WTV) period of time could be reset back to the smaller window. One embodiment of increasing may start with 10% of the WTV, then 20% at the next thread, 30% at the next thread, up to 90%, until a successful whereabouts is determined. After successful whereabouts determination, a reset to its original starting value is made.
- A semaphore accessed thread **1952** busy flag is used for indicating a certain thread is busy to prevent another **1952** thread from doing the same or similar work. Furthermore, other semaphore protected data for what work

is actually being performed by a thread can be informative to ensure that no thread **1952** starts for doing duplicated effort.

Useful data of statistics **14** may be appropriately accessed by thread(s) **1952** for dynamically controlling key variables of FIG. **26**B processing, such as the search f(WTV) time period, sort keys used, when to quit loop processing (e.g. on first successful whereabouts determination at block **2680**), surrounded-ness preferences, etc. This can dynamically change the FIG. **26**B logic from one thread to another for desired results.

FIG. **26**B continues processing through every WDR retrieved at block **2634**. An alternative embodiment will terminate processing after finding the first (which is highest priority data supported) successful triangulation at block **2682**.

FIG. **27** depicts a flowchart for describing a preferred embodiment of queue prune processing. Queue pruning is best done on an interim basis by threads which may insert to the queue being pruned. In an alternate embodiment, a background asynchronous thread will invoke FIG. **27** for periodic queue pruning to ensure no queue which can grow becomes too large. The Prune Queues procedure starts at block **2702** and continues to block **2704** where parameters passed by a caller for which queue(s) (WDR and/or CR) to prune are determined. Thereafter, if block **2706** determines that the caller wanted to prune the WDR queue **22**, block **2708** appropriately prunes the queue, for example discarding old entries using field **1100***b*, and processing continues to block **2710**. If block **2706** determines that the caller did not want to prune the WDR queue **22**, then processing continues to block **2710**. If block **2710** determines that the caller wanted to prune the CR queue **1990**, block **2712** appropriately prunes the queue, for example discarding old entries using field **2450***a*, and processing continues to block **2714**. If block **2710** determines that the caller did not want to prune the CR queue **1990**, then processing continues to block **2714**. Block **2714** appropriately returns to the caller.

The current design for queue **1980** does not require FIG. **27** to prune it. Alternative embodiments may add additional queues for similar processing. Alternate embodiments may use FIG. **27** like processing to prune queues **24**, **26**, or any other queue under certain system circumstances. Parameters received at block **2704** may also include how to prune the queue, for example when using different constraints for what indicates entry(s) for discard.

FIG. **28** depicts a flowchart for describing a preferred embodiment of MS termination processing. Depending on the MS, there are many embodiments of processing when the MS is powered off, restarted, rebooted, reactivated, disabled, or the like. FIG. **28** describes the blocks of processing relevant to the present disclosure as part of that termination processing. Termination processing starts at block **2802** and continues to block **2804** for checking any DLM roles enabled and appropriately terminating if any are found (for example as determined from persistent storage variable DLMV). Block **2804** may cause the termination of thread(s) associated with enabled DLM role(s) for DLM processing above (e.g. FIGS. **2**A through **9**B). Block **2804** may invoke API(s), disable flag(s), or terminate as is appropriate for DLM processing described above. Such terminations are well known in the art of prior art DLM capabilities described above. Block **2804** continues to block **2806**.

Blocks **2806** through **2816** handle termination of all processes/threads associated with the ILMV roles so there is no explicit ILMV check required. Block **2806** initializes an enumerated process name array for convenient processing reference of associated process specific variables described in FIG. **19**, and continues to block **2808** where the first member of the set is accessed for subsequent processing. The enumerated set of process names has a prescribed termination order for MS architecture **1900**. Thereafter, if block **2810** determines the process identifier (i.e. **19***xx*-PID such that **19***xx* is **1902**, **1912**, **1922**, **1932**, **1942**, **1952** in a loop iteration of blocks **2808** through **2816**) is greater than 0 (e.g. this first iteration of **1912**-PID>0 implies it is to be terminated here; also implies process **1912** is enabled as used in FIGS. **14**A, **28**, **29**A and **29**B), then block **2812** prepares parameters for FIG. **29**B invocation, and block **2814** invokes (calls) the procedure of FIG. **29**B to terminate the process (of this current loop iteration (**19***xx*)). Block **2812** prepares the second parameter in accordance with the type of **19***xx* process. If the process (**19***xx*) is one that is slave to a queue for dictating its processing (i.e. blocked on queue until queue entry present), then the second parameter (process type) is set to 0 (directing FIG. **29**A processing to insert a special termination queue entry to be seen by worker thread(s) for terminating). If the process (**19***xx*) is one that is slave to a timer for dictating its processing (i.e. sleeps until it is time to process), then the second parameter (process type) is set to the associated **19***xx*-PID value (directing FIG. **29**B to use in killing/terminating the PID in case the worker thread(s) are currently sleeping). Block **2814** passes the process name and process type as parameters to FIG. **29**B processing. Upon return from FIG. **29**B, block **2814** continues to block **2816**. If block **2810** determines that the **19***xx* process is not enabled, then processing continues to block **2816**. Upon return from FIG. **29**B processing, the process is terminated and the associated **19***xx*-PID variable is already set to 0 (see blocks **2966**, **2970**, **2976** and **2922**).

Block **2816** checks if all process names of the enumerated set (**19***xx*) have been processed (iterated) by blocks **2808** through **2816**. If block **2816** determines that not all process names in the set have been processed (iterated), then processing continues back to block **2808** for handling the next process name in the set. If block **2816** determines that all process names of the enumerated set were processed, then block **2816** continues to block **2818**.

Block **2818** destroys semaphore(s) created at block **1220**. Thereafter, block **2820** destroys queue(s) created at block **1218** (may have to remove all entries first in some embodiments), block **2822** saves persistent variables to persistent storage (for example to persistent storage **60**), block **2824** destroys shared memory created at block **1212**, and block **2826** checks the NTP use variable (saved prior to destroying shared memory at block **2824**).

If block **2826** determines NTP is enabled, then block **2828** terminates NTP appropriately (also see block **1612**) and processing continues to block **2830**. If block **2826** determines NTP was not enabled, then processing continues to block **2830**. Block **2828** embodiments are well known in the art of NTP implementations. Block **2828** may cause terminating of thread(s) associated with NTP use.

Block **2830** completes LBX character termination, then block **2832** completes other character **32** termination processing, and FIG. **28** processing terminates thereafter at block **2834**. Depending on what threads were started at block **1240**, block **2830** may terminate the listen/receive threads for feeding queue **26** and the send threads for sending data inserted to queue **24**. Depending on what threads were started at block **1206**, block **2832** may terminate the listen/receive threads for feeding queue **26** and the send threads for sending data inserted to queue **24** (i.e. other character **32** threads altered to cause embedded CK processing). Upon encounter of block

2834, the MS is appropriately terminated for reasons at set forth above for invoking FIG. 28.

With reference now to FIG. 29B, depicted is a flowchart for describing a preferred embodiment of a procedure for terminating a process started by FIG. 29A. When invoked by a caller, the procedure starts at block 2952 and continues to block 2954 where parameters passed are determined. There are two parameters: the process name to terminate, and the type of process to terminate. The type of process is set to 0 for a process which has worker threads which are a slave to a queue. The type of process is set to a valid O/S PID when the process worker threads are slave to a timer.

Thereafter, if block 2956 determines the process type is 0, then block 2958 initializes a loop variable J to 0, and block 2960 inserts a special termination request queue entry to the appropriate queue for the process worker thread to terminate. See FIG. 19 discussions for the queue inserted for which 19*xx* process name.

Thereafter, block 2962 increments the loop variable by 1 and block 2964 checks if all process prescribed worker threads have been terminated. Block 2964 accesses the 19*xx*-Max (e.g. 1952-Max) variable from shared memory using a semaphore for determining the maximum number of threads to terminate in the process worker thread pool. If block 2964 determines all worker threads have been terminated, processing continues to block 2966 for waiting until the 19*xx*-PID variable is set to disabled (e.g. set to 0 by block 2922), and then to block 2978 which causes return to the caller. Block 2966 uses a preferred choice of waiting described for blocks 2918 and 2920. The 19*xx* process (e.g. 1952) will have its 19*xx*-PID (e.g. 1952-PID) variable set at 0 (block 2922) when the process terminates. In some embodiments, the waiting methodology used at block 2966 may use the 19*xx*-PID variable, or may be signaled by the last terminating worker thread, or by block 2922.

If block 2964 determines that not all worker threads have been terminated yet, then processing continues back to block 2960 to insert another special termination request queue entry to the appropriate queue for the next process worker thread to terminate. Blocks 2960 through 2964 insert the proper number of termination queue entries to the same queue so that all of the 19*xx* process worker threads terminate.

Referring back to block 2956, if it is determined the process type is not 0 (i.e. is a valid O/S PID), then block 2968 inserts a special WDR queue 22 entry enabling a queue peek for worker thread termination. The reader will notice that the process termination order of block 2806 ensures processes which were slaves to the WDR queue 22 have already been terminated. This allows processes which are slaves to a timer to see the special termination queue entry inserted at block 2968 since no threads (which are slaves to queue) will remove it from queue 22. Thereafter, block 2970 waits until the 19*xx* process name (parameter) worker threads have been terminated using a preferred choice of waiting described for blocks 2918 and 2920. The 19*xx* process (e.g. 1902) will have its 19*xx*-PID (e.g. 1902-PID) variable set at 0 (block 2922) when the process terminates. In some embodiments, the waiting methodology used at block 2970 may use the 19*xx*-PID variable, or may be signaled by the last terminating worker thread, or by block 2922. Block 2970 also preferably waits for a reasonable timeout period in anticipation of known sleep time of the 19*xx* process being terminated, for cases where anticipated sleep times are excessive and the user should not have to wait for lengthy FIG. 28 termination processing. If the timeout occurs before the process is indicated to be terminated, then block 2970 will continue to block 2972. Block 2970 also continues to block 2972 when the process has successfully terminated.

If block 2972 determines the 19*xx* process did terminate, the caller is returned to at block 2978 (i.e. 19*xx*-PID already set to disabled (0)). If block 2972 determines the 19*xx* process termination timed out, then block 2974 forces an appropriate O/S kill to the PID thereby forcing process termination, and block 2976 sets the 19*xx*-PID variable for disabled (i.e. process 19*xx* was terminated). Thereafter, block 2978 causes return to the caller.

There are many embodiments for setting certain queue entry field(s) identifying a special queue termination entry inserted at blocks 2960 and 2968. Some suggestions: In the case of terminating thread(s) 1912, queue 26 insertion of a WDR preferably sets the MS ID field with a value that will never appear in any other case except a termination request (e.g. −100). In the case of terminating thread(s) 1902, 1922 and 1952, queue 22 insertion of a WDR preferably sets the MS ID field with a value that will never appear in any other case except a termination request (e.g. −100). In the case of terminating thread(s) 1942, queue 26 insertion of a WDR request preferably sets the MS ID field with a value that will never appear in any other case except a termination request (e.g. −100). In the case of terminating thread(s) 1932, queue 1980 insertion of a thread request queue record 2400 preferably sets field 2400*a* with a value that will never appear in any other case except a termination request (e.g. −100). Of course, any available field(s) can be used to indicate termination to particular thread(s)).

Terminating threads of processing in FIG. 29B has been presented from a software perspective, but there are hardware/firmware thread embodiments which may be terminated appropriately to accomplish the same functionality. If the MS operating system does not have an interface for killing the PID at block 2974, then blocks 2972 through 2976 can be eliminated for relying on a FIG. 28 invocation timeout (incorporated for block 2814) to appropriately rob power from remaining thread(s) of processing.

An ILM has many methods and systems for knowing its own location. LBX depends on MSs maintaining their own whereabouts. No service is required to maintain the whereabouts of MSs in order to accomplish novel functionality.

LBX: Permissions and Charters—Configuration

Armed with its own whereabouts, as well as whereabouts of others and others nearby, a MS uses charters for governing many of the peer to peer interactions. A user is preferably unaware of specificities of the layer(s) providing WDR interoperability and communications. Permissions 10 and charters 12 surface desired functionality to the MS user(s) without fully revealing the depth of features that could be made available. Permissions provide authentication for novel features and functionality, and to which context to apply the charters. However, some permissions can provide action(s), features, and functionality by themselves without a charter. It is preferred that LBX features and functionality be provided in the most elegant manner across heterogeneous MSs.

User configured permissions are maintained at a MS and their relevance (applicability) to WDRs that are being processed is determined. WDR processing events are recognized through being placed in strategic LBX processing paths of WDRs. For example, permissions govern processing of newly processed WDRs at a MS, regardless of where the WDR originated. A permission can provide at least one privilege, and may provide a plurality of privileges. A permission

is granted from a grantor identity to a grantee identity. Depending on what permissions are determined relevant to (i.e. applicable to) a WDR being processed (e.g. by accessing at least one field in the WDR), an action or plurality of actions which are associated with the permission can automatically occur. Actions may be as simple as modifying a setting which is monitored/used by an LBX application, or as complex as causing many executable application actions for processing. User configured charters are maintained at a MS and their relevance applicability) to WDRs that are being processed is determined, preferably in context of the same recognized events (i.e. strategic processing paths) which are used for determining relevance of permissions to WDRs. A charter consists of a conditional expression and can have an action or plurality of actions which are associated with the expression. Upon evaluating the expression to an actionable condition (e.g. evaluates to a Boolean true result), the associated action(s) are invoked. Charters can be created for a MS by a user of that MS, or by a user of another MS. Charters are granted similarly to permissions in using a grantor and grantee identity, therefore granting a charter is equivalent to granting a permission to execute the charter.

While some embodiments will provide disclosed features as one at a time implementations, a comprehensive architecture is disclosed for providing a platform that will survive LBX maturity. FIGS. **30**A through **30**E depict a preferred embodiment BNF (Backus Naur Form) grammar for permissions **10** and charters **12**. A BNF grammar is an elegant method for describing the many applicable derived subset embodiments of syntax and semantics in carrying out processing behavior. The BNF grammar of FIGS. **30**A through **30**E specifically describes:

- Prescribed command languages, such as a programming language, for encoding/representing permissions **10** and charters **12** (e.g. a Whereabouts Programming Language (WPL));
- Prescribed configuration in a Lex & Yacc processing of a suitable encoding;
- Prescribed XML encodings/representations of permissions **10** and charters **12**;
- Prescribed communications datastream encodings/representations of permissions **10** and charters **12**, such as in an ANSI encoding standard (e.g. X.409);
- Prescribed internalized encodings/representations of permissions **10** and charters **12**, for example in a data processing memory;
- Prescribed internalized encodings/representations of permissions **10** and charters **12**, for example in a data processing storage means;
- Prescribed database schemas for encoding/representing permissions **10** and charters **12**;
- Prescribed semantics of constructs to carry out permissions **10** and charters **12**;
- A delimited set of constructs for defining different representative syntaxes for carrying out permissions **10** and charters **12**; and
- Prescribed data processing of interpreters and/or compilers for internalizing a syntax for useful semantics as disclosed herein.

There are many embodiments (e.g. BNF grammar subsets) of carrying out permissions **10** and charters **12** without departing from the spirit and scope of the present disclosure. A particular implementation will choose which derivative method and system to implement, and/or which subset of the BNF grammars shall apply. Atomic elements of the BNF grammar (leaf nodes of the grammar tree) are identified within double quotes (e.g. "text string" implies the value is an atomic element in text string form). Atomic elements are not constructs which elaborate to other things and/or types of data.

FIGS. **30**A through **30**B depict a preferred embodiment BNF grammar **3002***a* through **3002***b* for variables, variable instantiations and common grammar for BNF grammars of permissions **10**, groups (e.g. data **8**) and charters **12**. Variables are convenient for holding values that become instantiated where appropriate. This provides a rich programming language and/or macro nature to the BNF grammar. Variables can be set with: a) a typed value (i.e. value of a particular data type (may be a list)); b) another variable for indirect referencing; c) a plurality of typed values; d) a plurality of variable references; or e) any combinations of a) through d). Variables can appear anywhere in the permissions or charters encodings. When variables are referenced by name, they preferably resolve to the name of the variable (not the value). When variables are referenced by their name with an instantiation operator (e.g. *), the variable is instantiated (i.e. elaborated/resolved) to assigned value(s). Instantiation also provides a macro (or function) ability to optionally replace subset(s) (preferably string replacements) of the variable's instantiated value with parameter substitutions. This enables customizably instantiating values (i.e. optionally, string occurrences in the value are replaced with specified matching parameters). An alternate embodiment to string substitution is for supporting numbers to be incremented, decremented, or kept as is, depending on the substitution syntax. For example:

*myVar(555++, 23−=4,888−−,200+=100)

This instantiation specifies that all occurrences of the string "555" should be incremented by 1 such that the first occurrence of "555" becomes "556", next occurrence of "555" becomes "557", and so on. Changing all occurrences of "555" to "556" is accomplished with the string substitution. This instantiation also specifies that all occurrences of the string "23" should be decremented by 4 such that the first occurrence of "23" becomes "19", next occurrence of "23" becomes "15", and so on. Changing all occurrences of "23" to "19" is accomplished with the string substitution. This instantiation also specifies that all occurrences of the string "888" should be decremented by 1 such that the first occurrence of "888" becomes "887", next occurrence of "888" becomes "886", and so on. Changing all occurrences of "888" to "887" is accomplished with the string substitution. This instantiation also specifies that all occurrences of the string "200" should be incremented by 100 such that the first occurrence of "200" becomes "300", next occurrence of "200" becomes "400", and so on. Changing all occurrences of "200" to "300" is accomplished with the string substitution.

Preferably, when a variable is set to another variable (e.g. a=b), an instantiation of the variable (i.e. *a) equals the variable b, not b's value (i.e. *(*a)=b's value). If the variable b is set to a variable c (e.g. b=c) in the example, and the variable a is set to the variable b as already described (past or future, prior to instantiation), and c was set (i.e. c=2) to the value 2 (past or future, prior to instantiation), then the preferred embodiment requires three (3) instantiations of variable a to get to the value assigned to variable c (e.g. *(*(*a)))=2). Instantiation of variable a (e.g. *a) preferably corresponds to a level of "peeling back" through the hierarchy of variable assignments if one exists. Alternative embodiments will allow a single instantiation of a variable to get through any number of indirect variable assignments for the first encountered value in the indirect chain value (e.g. *a=2) at the time of instantiation. Either semantic may have useful features from a programming standpoint. Over-instantiating

(e.g. *(*c)=error) should cause an error. An assigned value is the leaf node in peeling back with instantiations.

The BNF Grammar "null" is an atomic element for no value. In a syntactic embodiment, a null value may be a special null character (e.g. Ø). The History construct is preferably used to track when certain constructs were created and last modified. An alternative embodiment will track all construct changes to LBX history **30** for later human, or automated, processing audit.

Grammar **3002***b* "system type" is an atomic element (atomic elements are not constructs which elaborate to other things; atomic elements are shown delimited in double quotes) generalized for the type of MS (e.g. PDA, cell phone, laptop, etc). Other embodiments will provide more detail to the type of MS (e.g. iPhone, Blackberry Pearl, Nextel i845, Nokia 741, etc). ID is an identity construct of the present disclosure for identifying a MS, a user, a group, or any other entity for which to associate data and/or processing. IDType provides the type of ID to support a heterogeneous identifying grammar. An identity (i.e. ID [IDType]) can be directly associated to a MS (e.g. MS ID), or may be indirectly associated to a MS (e.g. user ID or group ID of the MS). Indirect identity embodiments may assume an appropriate lookup for mapping between identities is performed to get one identity by looking up another identity. There may be multiple identities for a MS. Identities, by definition, provide a collective handle to data. For example, an email sender or recipient is an example of an identity ("logical handle") which can be associated to a user identity and/or MS identity and/or group identity. A sender, source, recipient, and system parameter in some atomic commands presented below is any of the variety of types of identities.

Address elements of "ip address" and "SNA address" are examples of logical addresses, but are mentioned specifically anyway. ID, IDType and Address construct atomic elements (as elaborated on Right Hand Side (RHS)) are self explanatory. The TimeSpec construct is one of various kinds of "date/time stamp" or "date/time period" atomic elements. In a syntactic embodiment, date/time stamps are specified with prefixed character(s) and a time format such as xYYYYMMDDHHMMSS.12 . . . J (J=# places to right of decimal point, such that 1=is the one tenth (1/10) second place, two=the one hundredth (1/100) second place, etc). The first character(s) (i.e. x) clarify the date/time stamp information.

- >20080314 indicates "in effect if current date/time after Mar. 14, 2008;
- >=20080314 indicates "in effect if current date/time on or after Mar. 14, 2008;
- <200803142315 indicates "in effect if current date/time prior to Mar. 14, 2008 at 11:15 PM;
- <=200803142315 indicates "in effect if current date/time on or prior to Mar. 14, 2008 at 11:15 PM; and
- =20080314231503 indicates "in effect if current date/time matches Mar. 14, 2008 at 11:15:03 PM.

Date/time periods may have special leading characters, just as described above (which are also periods). When using the date/time format, the granulation of the date/time stamp is a period of time.

- 20080314 indicates "in effect if current date/time during Mar. 14, 2008;
- 200803142315 indicates "in effect if current date/time during Mar. 14, 2008 at 11:15 PM (any time during that minute); and
- 20080314231503 indicates "in effect if current date/time during Mar. 14, 2008 at 11:15:03 PM (any time during that second).

Date/time periods can also be specified with a range using a colon such as 20080314:20080315 (Mar. 14, 2008 through Mar. 15, 2008). A date/time period can be plural such as 20080314:20080315, 2008031712:2008031823 (i.e. multiple periods) by using a comma.

FIG. **30**C depicts a preferred embodiment BNF grammar **3034** for permissions **10** and groups (of data **8**). The terminology "permissions" and "privileges" are used interchangeably in this disclosure. However, the BNF grammar shows a permission can provide one privilege, or a plurality of privileges. There are a massive number (e.g. thousands) of values for "atomic privilege for assignment" (i.e. privileges that can be assigned from a grantor to a grantee) in grammar **3034**. Few examples are discussed below. This disclosure would be extremely lengthy to describe every privilege. The reader can determine a minimum set of LBX privileges (permissions) disclosed as: Any configurable privilege granted by one identity to another identity that can limit, enable, disable, delegate, or govem actions, feature(s), functionality, behavior(s), or any subset(s) thereof which are disclosed herein. Every feature disclosed herein, or feature subset thereof, can be managed (granted and enforced) with an associated privilege. Privileges may be used to "turn on" a feature or "turn off" a feature, depending on various embodiments.

There are two (2) main types of permissions (privileges): semantic privileges which on their own enable LBX features and functionality; and grammar specification privileges which enable BNF grammar specifications. Semantic privileges are named, anticipated by applications, and have a semantic meaning to an application. Semantic privileges are variables to applications whereby values at the time of an application checking the variable(s) determine how the application will behave. Semantic privileges can also have implicit associated action(s). Grammar specification privileges are named, anticipated by charter parser implementation, and indicate what is, and what is not, permitted when specifying a charter. Grammar specification privileges are variables to charter parsing whereby values at the time of charter parse logic checking the variable(s) determine whether or not the charter is valid (i.e. privileged) for execution. Impersonation is not directly defined in the BNF grammar of charters, and is therefore considered a semantic privilege.

The "MS relevance descriptor" atomic element is preferably a binary bit-mask accommodating all anticipated MS types (see "system type"). Each system type is represented by a bit-mask bit position wherein a bit set to 1 indicates the MS type does participate with the privilege assigned, and a bit set to 0 indicates the MS type does not participate with the privilege assigned. This is useful when MSs do not have equivalent capabilities thereby limiting interoperability for a particular feature governed by a privilege. When the optional MSRelevance construct is not specified with a privilege, the preferred default is assumed relevance for all MSs (i.e. =all bits set to 1). An alternate embodiment will make the default relevant for no MSs (i.e. =all bits set to 0). Privilege codes (i.e. syntactical constants equated to an "atomic privilege for assignment" description) are preferably long lived and never changing so that as new LBX privileges are introduced (i.e. new privileges supported), the old ones retain their values and assigned function, and operate properly with new software releases (i.e. backwards compatible). Thus, new constants (e.g. \lbxall=privilege for allowing all LBX interoperable features) for "atomic privilege for assignment" should be chosen carefully.

Grants are used to organize privileges in desired categories and/or sub-categories (e.g. organization name, team name, person name, etc and then privileges for that particular grant

name). A grant can be used like a folder. Grants provide an hierarchy of tree branch nodes while privileges are leaf nodes of the grant privilege tree. There are many types of privileges. Many are categorized for configuring charter conditions and charter actions, and some can be subsets of others, for example to have an overall category of privileges as well as many subordinate privileges within that category. This facilitates enabling/disabling an entire set with a single configuration, or enabling/disabling certain privileges within the set. This also prevents forcing a user to define Grants to define privilege categories. BNF grammar **3034** does not clarify the Privilege construct with a parameter for further interpretation, however some embodiments will incorporate an optional Parameters specification:

Privilege="atomic privilege for assignment" [Parameters] [MSRelevance][TimeSpec] [Description] [History] |Varinstantiations

In such embodiments, Parameters preferably resolves to the Parameters construct of FIG. **30**E for clarifying how to apply a particular privilege. Parameters, if used for privileges, have meaning within the context of a particular privilege. Some examples of semantic privileges (i.e. "atomic privilege for assignment") that can be granted from a grantor identity (ID/IDType) to a grantee identity (ID/IDType) include:

- Impersonate: allows the grantee to perform MS administration of grantor (alternate embodiments will further granulate to a plurality of impersonate privileges for each possible type, or target, of administration);
- LBX interoperable: allows overall LBX interoperability (all or none);
- View nearby status: enables determining if nearby each other;
- View whereabouts status: enables determining whereabouts (e.g. on a map);
- View Reports: enables viewing statistics and/or reports; This privilege is preferably set with a parameter for which statistics and/or which reports; An alternate embodiment will have individual privileges for each type of statistic and/or report;
- View Historical Report: enables viewing history information (e.g. routes); This privilege is preferably set with a parameter for which history information; An alternate embodiment will have individual privileges for each type of history information;
- Set Geofence arrival alert: allows an action for alerting based on arrival to a geofenced area; This privilege may be set with parameter(s) for which eligible area(s) to define geofences; An alternate embodiment will have individual privileges for each area(s);
- Set Geofence departure alert: allows an action for alerting based on departure from a geofenced area; This privilege may be set with parameter(s) for which eligible area(s) to define geofences; An alternate embodiment will have individual privileges for each area(s);
- Set nearby arrival alert: allows an action for alerting based on arrival to being nearby; This privilege may be set with a parameter for quantifying amount nearby;
- Set nearby departure alert: allows an action for alerting based on departure from being nearby; This privilege may be set with a parameter for quantifying amount nearby;
- Set Geofence group arrival alert: allows an action for alerting based on a group's arrival to a geofenced area; This privilege may be set with parameter(s) for which groups or MSs apply;
- Set Geofence group departure alert: allows an action for alerting based on a group's departure from a geofenced area; This privilege may be set with parameter(s) for which groups or MSs apply;
- Set nearby group arrival alert: allows an action for alerting based on a group's arrival to being nearby; This privilege may be set with parameter(s) for quantifying amount nearby, and/or which groups or MSs apply;
- Set nearby group departure alert: allows an action for alerting based on a group's departure from being nearby; This privilege may be set with parameter(s) for quantifying amount nearby, and/or which groups or MSs apply;
- Set Situational Location (as defined in U.S. Pat. Nos. 6,456,234; 6,731,238; 7,187,997; U.S. PTO Publication 2006/0022048 (Johnson)) arrival alert: allows an action for alerting based on arrival to a situational location; This privilege may be set with parameter(s) for one or more situational location(s) defined;
- Set Situational Location (as defined in U.S. Pat. Nos. 6,456,234; 6,731,238; 7,187,997; U.S. PTO Publication 2006/0022048 (Johnson)) departure alert: allows an action for alerting based on departure from a situational location; This privilege may be set with a parameter(s) for one or more situational location(s) defined;
- Set Situational Location (as defined in U.S. Pat. Nos. 6,456,234; 6,731,238; 7,187,997; U.S. PTO Publication 2006/0022048 (Johnson)) group arrival alert: allows an action for alerting based on a group's arrival to a situational location; This privilege may be set with parameter(s) for one or more situational location(s) defined, and/or which groups or MSs apply;
- Set Situational Location (as defined in U.S. Pat. Nos. 6,456,234; 6,731,238; 7,187,997; U.S. PTO Publication 2006/0022048 (Johnson)) group departure alert: allows an action for alerting based on a group's departure from a situational location; This privilege may be set with parameter(s) for one or more situational location(s) defined, and/or which groups or MSs apply;
- Allow action monitoring: allows condition for the monitoring of certain action(s); This privilege may be set with parameter(s) for which action(s) to be monitored;
- Accept service routing: enables being a service routing system; This privilege may be set with parameter(s) for which service(s) to route;
- Allow whereabouts monitoring (i.e. any WDR **1100** fields): allows condition for the monitoring of certain whereabouts; This privilege may be set with parameter(s) for which area(s) where whereabouts can be monitored; Another embodiment will define a specific privilege for each field and/or subfield of a WDR **1100** (e.g. speed monitoring (e.g. field **1100***h*));
- Service informant utilization (includes derived subsets for how to be used; e.g. log for me all successful detections (or particular types) by the remote MS of interest);
- Strip out WDR information inbound, outbound, and/or prior to be inserting to queue **22**: these types of privileges may also affect what charters can and cannot do;
- Support certain types of service informant code processing, for example for carpool collaboration;
- Participate in parking lot search functionality; this privilege may be set with parameter(s) for which parking lots apply;
- Be a candidate peer service target for any particular application, types of applications, or all applications, or for certain MSs, certain groups, or combinations of any of these (parameter(s) may be specified);
- Participate in LN-expanse as a master MS, for example to maintain a database of historical MSs in the vicinity, or

a database of identity mappings (e.g. users to MSs; (parameter(s) may be specified);

Keep track of hotspot history;

Provide service propagation for any particular application, types of applications, or all applications, or for certain MSs, certain groups, or combinations of any of these (parameter(s) may be specified);

Enable automatic call forwarding functionality when within proximity to a certain phone, for example to route a wireless call to a nearby wired line phone; this privilege may be set with parameter(s) for which phones or phone numbers participate;

Enable configuration of deliverable content that can be delivered in a peer to peer manner to a MS in the vicinity, using any data type, size, location, or other characteristic to be a unique privilege; parameter(s) may be specified to qualify this;

A privilege for any functionality or feature disclosed herein;

Any subordinate privilege of above, or of any functionality or feature disclosed herein;

Any parent privilege of above, or of any functionality or feature disclosed herein; and/or

Any privilege combination of above, or of any functionality or feature disclosed herein.

Grammar specification privileges can enable/disable permitted specifications of certain charter terms, conditions, actions, or any other charter aspect. Some examples of grammar specification privileges (i.e. "atomic privilege for assignment") that can be granted from a grantor identity (ID/IDType) to a grantee identity (ID/IDType) include:

Accept autodial #: allows an action for sending a speed dial number;

Accept web link: allows an action for sending a hyper link;

Accept email: allows an action for sending an email;

Accept SMS msg: allows an action for sending an SMS message;

Accept content: allows an action for sending a content of any type;

Accept broadcast email: allows an action for sending a broadcast email;

Accept broadcast SMS msg: allows an action for sending a broadcast SMS message;

Accept indicator: allows an action for sending an indicator;

Accept invocation: allows an action for invoking (optionally with parameters for which executable and parameters to it) an executable (application, script, command file, or any other executable); Alternate embodiments will have specific privileges for each type of executable that may be invoked);

Accept file: allows an action for sending a file or directory;

Accept semaphore control: allows an action for setting or clearing a semaphore; This privilege is preferably set with a parameter for which semaphore and what to do (set or clear);

Accept data control: allows an action for access, storing, alerting, or discarding data (alternate embodiments will further granulate to a plurality of data control privileges for each data control type (access, store, alter, discard, etc); This privilege may be set with parameter(s) for which data and what to do;

Accept database control: allows an action for access, storing, alerting, or discarding database data (alternate embodiments will further granulate to a plurality of data control privileges for each data control type (access, store, alter, discard, etc); This privilege may be set with parameter(s) for which database data and what to do;

Accept file control: allows an action for access, storing, alerting, or discarding file/directory path data (alternate embodiments will further granulate to a plurality of data control privileges for each data control type (access, store, alter, discard, etc); This privilege may be set with parameter(s) for which directory or file path(s) and what to do;

Allow profile match comparison: allows condition for the monitoring of certain profile(s); This privilege may be set with a parameter(s) for which profile(s) can be monitored/compared; An alternate embodiment will define a specific privilege for each ProfileMatch type;

Allow interest match comparison: allows condition for the monitoring of interests; This privilege may be set with parameter(s) for which interests can be monitored/compared; An alternate embodiment will define a specific privilege for each interest candidate;

Allow filters match comparison: allows condition for the monitoring of filters; This privilege may be set with parameter(s) for which filters can be monitored/compared; An alternate embodiment will define a specific privilege for each filter candidate;

Allow movement monitoring: allows condition for the monitoring of movement; This privilege may be set with parameter(s) for quantifying how much movement, and/or how long for lack of movement (an alternate embodiment will define distinct privileges for each movement monitoring type);

Allow application use monitoring: allows condition for the monitoring of application usage; This privilege may be set with parameter(s) for specifying which application(s) to monitor, and/or how long for usage of the application(s); Another embodiment specifies which aspect of the application is to be monitored (e.g. data, DB data, semaphore, thread/process invoke or terminate, file/directory data, etc);

Allow invocation monitoring: allows an action for monitoring application(s) used (optionally with parameter(s) for which application/executable); Alternate embodiments will have specific privileges for each application or executable of interest;

Allow application termination monitoring: allows condition for monitoring application(s) terminated (optionally with parameter(s) for which application/executable); Alternate embodiments will have specific privileges for each application or executable of interest;

Allow file system monitoring: allows condition for monitoring a file or directory; This privilege may be set with parameter(s) for specifying which path(s) to monitor, and/or what to monitor for, and how long for absence or removal of the path(s);

Allow semaphore monitoring: allows condition for monitoring a semaphore; This privilege may be set with parameter(s) for specifying which semaphore(s) to monitor, and/or what to monitor for (clear or set);

Allow data monitoring (file or directory): allows condition for monitoring data; This privilege may be set with parameter(s) for specifying which data to monitor, and/or what value to monitor for (charter condition like a debugger watch);

Allow data attribute monitoring (file or directory): allows condition for monitoring data attribute(s); This privilege may be set with parameter(s) for specifying which data attributes (e.g. chmod or attrib or extended attributes) to monitor, and/or what value to monitor for (charter condition like a debugger watch);

Allow database monitoring: allows condition for monitoring database data; This privilege may be set with parameter(s) for specifying which database data to monitor, and/or what value to monitor for (like a database trigger);

Allow sender monitor: allows condition for monitoring sender information; This privilege may be set with parameter(s) for specifying which sender address(es) to monitor email or SMS messages from (may have separate privileges for each type of distribution);

Allow recipient monitor: allows condition for monitoring recipient information; This privilege may be set with parameter(s) for specifying which recipient address(es) to monitor email or SMS messages to (may have separate privileges for each type of distribution);

Allow "modification" instead of "monitor"/"monitoring" for each monitor/monitoring privilege described above;

Allow focused title bar use: allows using the focused title bar for alerting;

A privilege for any BNF grammar atomic command, atomic operand, parameter(s), parameter type, atomic operator, or underlying action performed in a charter herein;

Any subordinate privilege of above, or of any functionality or feature disclosed herein;

Any parent privilege of above, or of any functionality or feature disclosed herein; and/or

Any privilege combination of above, or of any functionality or feature disclosed herein.

While the Grantor construct translates to the owner of the permission configuration according to grammar **3034**, impersonation permits a user to take on the identity of a Grantor for making a configuration. For example, a group by its very nature is a form of impersonation when a single user of the group grants permissions from the group to another identity. A user may also impersonate another user (if has the privilege to do so) for making configurations. In an alternative embodiment, grammar **3034** may include means for identifying the owner of the permission(s) granted. Group constructs provide means for collections of ID constructs, for example for teams, departments, family, whatever is selected for grouping by a name (atomic element "group name"). The impersonation privilege should be delegated very carefully in the preferred embodiment since the BNF grammar does not carry owner information except through a History construct use.

The Grantor of a privilege is the identity wanting to convey a privilege to another identity (the Grantee). The Grantee is the identity becoming privileged by administration of another identity (the Grantor). There are various embodiments for maintaining privileges, some embodiments having the side affect of increasing, or decreasing, the palette of available privileges for assignment. Privilege/Permission embodiments include:

1) Administrated privileges are maintained and enforced at the Grantor's MS. As privileged Grantee WDR information is detected at the Grantor's MS, or as Grantor WDR information is detected at the Grantor's MS: the appropriately privileged Grantee is provided with LBX application features at their (Grantee) MS in accordance with the privileges granted;

2) Administrated privileges are maintained and enforced at the Grantor's MS, but are also communicated to the Grantee's MS for being used by the Grantee for informative purposes. As privileged Grantee WDR information is detected at the Grantor's MS, or as Grantor WDR information is detected at the Grantor's MS: the appropriately privileged Grantee is provided with LBX application features at their (Grantee) MS in accordance with the privileges granted;

3) Administrated privileges are maintained at the Grantor's MS for administration purpose, but are used for governing features/processing at a Grantee MS. Privileges are appropriately communicated to a Grantee MS for WDR information processing, such that as Grantor WDR information is detected at the Grantee MS, the Grantee is provided with LBX application features at their (Grantee) MS in accordance with the privileges granted; and/or

4) Privileges are stored at both the Grantor's MS and the Grantee's MS for WDR information processing including any combination of #1 through #3 above (i.e. WDR information processing at each MS provides LBX features benefiting the Grantor and/or Grantee).

5) See FIG. **49**A discussions for some of the permission/privilege assignment considerations between a Grantor identity and a Grantee identity.

FIGS. **30**D through **30**E depict a preferred embodiment BNF grammar **3068***a* through **3068***b* for charters. Charters embody conditional events to be monitored and the actions to cause when those events occur. Notice there is still a Grantee and Grantor construct in charters, even in the face of having privileges for governing the charters. Grantor and Grantee constructs used in charters have to do with granting the permission/privilege to enable charters at a particular MS. Once they are enabled at a MS, permissions/privileges of grammar **3034** may be used to govern how the charters process.

It is important to note the context of terminology use "Grantor" and "Grantee" appears in, since they are similarly used in context of charters versus permissions. In both cases there is an acceptance/authentication/configuration granted by a Grantor to a Grantee. A permission Grantor grants a privilege to a Grantee. A charter Grantor grants a privilege to enable a Grantee's charters (may be at the mercy of privileges in the preferred embodiment). The Grantee construct in charters translates to the owner/creator/maintainer identity of the charter configuration according to grammar **3068***a* and **3068***b*, and the Grantor construct translates to an identity the Grantee has created the charter for, but does not necessarily have the privilege to do so, or does not necessarily have the privilege for any subset of processing of the charter. Privileges preferably govern whether charters are in effect, and how they are in effect. An alternative embodiment will activate (make in effect) a charter by granting it from one identity to another as shown in grammar **3068***a*. A charter consists of a conditional expression and can have an action or plurality of actions which are associated with the conditional expression. Upon evaluating the expression to an actionable condition (e.g. evaluates to a Boolean true result), the associated action(s) are invoked.

Impersonation permits a user to take on the identity of a Grantee for making a configuration. For example, a group by its very nature is a form of impersonation when a single user of the group administrates charters for the group. A user may also impersonate another user (if has the privilege to do so) for making configurations. In an alternative embodiment, grammar **3068***a* and **3068***b* may include means for identifying the owner of the charters administrated. The impersonation privilege should be delegated very carefully in the preferred embodiment since the BNF grammar does not carry owner information except through a History construct use.

The Grantee of a charter is the identity (e.g. creates and owns the charter) wanting to have its charters processed for another identity (the Grantor). The Grantor is the identity

targeted for processing the administrated charter(s) created by the Grantee. The terminology "Grantor" and "Grantee" will become reversed (to match privilege assignments) in an embodiment which grants charters like privileges. There are various embodiments for maintaining charters, some embodiments having the side affect of increasing, or decreasing, the palette of available charter processing deployed. Charter embodiments include:

6) Administrated charters are stored at the Grantee's (the administrator's) MS. As privilege providing Grantor WDR information is detected at the Grantee's MS, the Grantee is provided with LBX application charter processing at his (Grantee) MS, preferably in accordance with privileges defined as described in #1 through #5 above;

7) Administrated charters are maintained at the Grantee's (the administrator's) MS, but are communicated to the Grantor's MS for being used for informative purposes. As privilege providing Grantor WDR information is detected at the Grantee's MS, the Grantee is provided with LBX application charter processing at his (Grantee) MS, preferably in accordance with privileges defined as described in #1 through #5 above;

8) Administrated charters are maintained at the Grantee's MS for administration purpose, but are used for processing at the Grantor MS. Charters are appropriately communicated to the Grantor MS for WDR information processing, such that as Grantor WDR information is detected at the Grantor MS, the Grantee is provided with LBX application features for processing at the Grantor's MS, preferably in accordance with privileges defined as described in #1 through #5 above. Also, as Grantee WDR information is detected at the Grantor's MS, the Grantee is provided with LBX application charter processing at his (Grantee) MS, preferably in accordance with privileges defined as described in #1 through #5 above; and/or

9) Charters are maintained at both the Grantor's MS and the Grantee's MS for WDR information processing, including any combination of #6 through #8 above (i.e. WDR information processing at each MS provides LBX features benefiting the Grantor and/or the Grantee).

10) See FIG. **49**B discussions for some of the charter assignment considerations between a Grantee identity and a Grantor identity.

Grammar **3068***a* "and" and "or" are atomic elements for CondOp operators. In a syntactic embodiment, "and" and "or" may be special characters (e.g. &, |, respectively). Grammar **3068***a* Value elaboration "atomic term" (RHS) is an atomic element for a special type of term that can be used in a condition specification, such as:

My MS location (e.g. \loc_my): preferred embodiment resolves to field **1100***c* from the most recent WDR which describes this MS (i.e. the MS of atomic term evaluation processing); WTV may be used to determine if this is of use (if not, may return a null, cause a failure in a conditional match, or generate an error);

A specified MS, or group, mobile location (e.g. \locByL__−30.21,−97.2=location at the specified latitude and longitude (ensure no intervening blanks): preferred embodiment resolves to a specified location comparable to a WDR field **1100***c*, not necessarily in the same format or units used as field **1100***c* (i.e. converted appropriately for a valid comparison when used). There are many different formats and units that can be specified here with a unique syntax;

A specified MS, or group, situational location (e.g. \slByL__−30.21,−97.2;1050F=situational location at the specified latitude, longitude and elevation in feet (ensure no intervening blanks): preferred embodiment resolves to specified situational location comparable to applicable WDR fields, not necessarily in the same format or units used (i.e. converted appropriately for valid comparison(s) when used). See U.S. Pat. No. 6,456,234 (Johnson) for the definition of a situational location that can be specified. A reasonable syntax following the leading escape character and "sl" prefix should be used; this example assumes an anticipated order (lat, long, elevation); One embodiment also assumes an order for other situational location criteria wherein a semicolon (;) delimits data (i.e. use ";" to show lack of data at anticipated position (e.g. \slByL__−30.21,−97.2;;;;56); Another embodiment uses descriptors to indicate which data is being described so any order can be specified (e.g. \slByL_lat=−30.21,lon=−97.2;elev=1050F). There are many different formats, fields and units that can be specified here with a unique syntax;

My current MS mobile location (e.g. \loc_my): same as described above;

A current MS, or group, mobile location (e.g. \locByID_Larry=location of MS with id Larry, \locG_dept78=location of members of the group dept78): preferred embodiment resolves to a location associated with an identifier. Preferably, queue **22** is accessed first for the most recent occurrence of a WDR matching the identifier(s). An alternate embodiment additionally searches LBX history **30** if not found elsewhere. In one embodiment, an averaged location is made for a group identifier using locations of the identifiers belonging to the group, otherwise a group containing MSs with different locations causes a false condition when used in an expression, or alternatively cause an error. This is preferably used to compare locations of WDRs from a plurality of different MSs without requiring a value to be surfaced back to the expression reference;

A current MS, or group, situational location (e.g. \slByID_Larry=situational location of MS with id Larry, \slG_dept78=situational location of members of the group dept78): preferred embodiment resolves to a situational location associated with an identifier. Preferably, queue **22** is accessed first for the most recent occurrence of a WDR matching the identifier(s). An alternate embodiment additionally searches LBX history **30** if not found elsewhere. In one embodiment, an averaged situational location is made for a group identifier using locations of the identifiers belonging to the group, otherwise a group containing MSs with different locations causes a false condition when used in an expression, or alternatively cause an error. This is preferably used to compare situational locations of WDRs from a plurality of different MSs without requiring a value to be surfaced back to the expression reference;

Last application used (e.g. \appLast): preferably resolves to an application reference (e.g. name) which can be successfully compared to a MS operating system maintained reference for the application (e.g. as maintained to LBX history) that was last used by the MS user (e.g. embodiments for last focused, or last used that had user input directed to it). One embodiment implements only known PRR applications using field **5300***a* and/or **5300***b* for the reference (See FIGS. **53** and **55**A);

Last application context used (e.g. \appLastCtxt): preferably resolves to an application context reference which can be successfully compared to a MS operating system context maintained for comparison to LBX history. One embodiment implements only known PRR applications using field **5300***a* and/or **5300***b* for the application reference (See FIGS. **53** and **55**A), and saved user input for the context of when the application was focused. Another embodiment incorporates the system and methods of U.S. Pat. No. 5,692,143 ("Method and system for recalling desktop states in a data processing system", Johnson et al) to maintain application contexts to history;

Application in use (e.g. \appLive): preferably resolves to an application reference (e.g. name) which can be successfully compared to a MS operating system maintained reference for the application (e.g. as maintained to LBX history) that may or may not be running (active) on the MS. One embodiment implements only known PRR applications using field **5300***a* and/or **5300***b* for the reference (See FIGS. **53** and **55**A);

Application context in use (e.g. \appLiveCtxt): preferably resolves to an application context reference which can be successfully compared to a MS operating system context maintained for comparison. One embodiment implements only known PRR applications using field **5300***a* and/or **5300***b* for the application reference (See FIGS. **53** and **55**A), and saved user input for the current context of the application (e.g. maintained to LBX history). Another embodiment incorporates the system and methods of U.S. Pat. No. 5,692,143 ("Method and system for recalling desktop states in a data processing system", Johnson et al) to maintain application contexts;

Application active (e.g. \appLive): same as application in use above;

Application context active (e.g. \appLiveCtxt): same as application context in use above;

Current MS system date/time (e.g. \timestamp); preferably resolves to the MS date/time from the MS system clock interface for a current date/time stamp;

Particular LBX maintained statistical value (e.g. \st_statisticName wherein statisticName is the name of the statistic): preferably resolves to the referenced statistic name of statistics **14**. There are potentially hundreds of statistics maintained for the MS;

MS ID of MS hosting atomic term (e.g. \thisms; alternate embodiments support ID and IDType grammar rules): preferably resolves to the identifier of the MS where the atomic term is being resolved; and/or

Most current WDR field of \thisMS (e.g. \fldname); fldname is identical to WDR in-process field names which can reference any field, subfield, set, subset, or derived data/information of a WDR in process (i.e. _fldname, _I_fldname, _O_fldname). The difference here is that the most recent WDR (e.g. of queue **22**) for \thisMS is accessed, rather than an in-process WDR. The leading backslash indicates to reference the most recent WDR for \thisMS. In some embodiments, the WTV is accessed and an error is produced for \fldname references that reference stale WDR information.

Preferably, a convenient syntax using a leading escape character refers to an atomic term (e.g. \loc_my=My MS location). When used in conjunction with other conditions, an "atomic term" provides extraordinary location based expressions. Other Grammar **3068***a* atomic elements are described here: "Any WDR **1100** field, or any subset thereof" is self explanatory; "Any Application data field, or any subset thereof" is an atomic element for any semaphore, data, database data, file/directory data, or any other reference-able data of a specified application; "number" is any number; "text string" is any text string; "True" is a Boolean representing true; "False" is a Boolean representing false; "typed memory pointer" is a pointer to memory location (of any memory or storage described for FIG. **1**D) containing a known type of data and length; "typed memory value" is a memory location (of any memory or storage described for FIG. **1**D) containing a known type of data and length; "typed file path" is a file path location (of any memory or storage described for FIG. **1**D) containing a known type of data and length; "typed file path and offset" is a file path location (of any memory or storage described for FIG. **1**D) and an offset therein (e.g. byte offset) for pointing to a known type of data and length; "typed DB qualifier" is a database data path (of any memory or storage described for FIG. **1**D) for qualifying data in a database (e.g. with a query, with a identity/table/row/column qualifier, or other reasonable database qualifying method).

WDRTerm provides means for setting up conditions on any WDR **1100** field or subfield that is detected for WDR(s):

Inserted by FIG. **2**F processing (e.g. received from other MSs, or created by the hosting MS); and/or

Sent/communicated outbound from a MS; and/or

Received/communicated inbound to a MS.

An alternate BNF grammar embodiment qualifies the "Any WDR **1100** field, or any subset thereof" atomic element with an operator for which of the three MS code paths to check WDR field conditions (e.g. Operators of "OUTBOUND" and "INBOUND", denoted by perhaps a syntactical O and I, respectively). Absence of an operator can be assumed for checking WDRs on FIG. **2**F insert processing. Such embodiments result in a BNF grammar WDRTerm definition of:

WDRTerm=[WDRTermOp] "Any WDR **1100** field, or any subset thereof" [Description] [History] |Varinstantiate

WDRTermOp="inbound"|"outbound"

Yet another embodiment will allow combination operators for qualifying a combination of any three MS code paths to check.

AppTerm provides means for setting up conditions on data of any application of an MS, for example to trigger an action based on a particular active call during whereabouts processing. A few AppTerm examples are any of the following:

Any phone application data record data (e.g. incoming call(s), outgoing call(s), active call(s), caller id, call attributes, etc)

Any email/SMS message application data record data (e.g. mailbox attributes, message last sent, message last received, message being composed, last type of message sent, last type of message received, attribute(s) of any message(s), etc)

Any address book application data record data (e.g. group(s) defined, friend(s) defined, entry(s) defined and any data associated with those, etc)

Any calendar application data record data (e.g. last scheduled entry, most recently removed entry, number of entries per time period(s), last scheduled event attendee(s), number of scheduled events for specified qualifier, next forthcoming appointment, etc)

Any map application data record data; and/or

Any other application data record data of a MS.

Grammar **3068***b* completes definition of grammar rules for charters. The Invocation construct elaborates to any of a variety of executables, with or without parameters, including Dynamic Link Library (DLL) interfaces (e.g. function), post-compile linked interfaces (e.g. function), scripts, batch files, command files, or any other executable. The invoked inter-

face should return a value, preferably a Boolean (true or false), otherwise one will preferably be determined or defaulted for it. The Op construct contains atomic elements (called atomic operators) for certain operators used for terms to specify conditions. In syntactical embodiments, each atomic operator may be clarified with a not modifier (i.e. !). For example, "equal to" is "=" and "not equal to" is "!=". Those skilled in the art recognize which atomic operator is contextually appropriate for which applicable terms (see BNF grammar **3068***a*). There are many reasonable syntactical embodiments for atomic operators, with at least:

=: equal to;
!=: not equal to;
>: greater than;
!>: not greater than;
>=: greater than or equal to;
!>=: not greater than or equal to;
<: less than;
!<: not less than;
<=: less than or equal to;
!<=: not less than or equal to;
^: in;
!^: not in;
^^: was in;
!^^: was not in;
@: at;
!@: not at;
@@: was at;
!@@: was not at;
$(range): in vicinity of (range=distance (e.g. 10 F=10 Feet));
!$(range): not in vicinity of (range=distance (e.g. 1 L=1 Mile));
>$(range): newly in vicinity of;
!>$(range): not newly in vicinity of;
$>(range): departed from vicinity of;
!$>(range): not departed from vicinity of;
(spec)$(range)
: recently in vicinity of (spec=time period (e.g. 8 H=in last 8 hours));
(spec)!$(range)
: not recently in vicinity of (spec=time period (e.g. 8 H=in last 8 hours));
(spec)$$(range)
: recently departed from vicinity of (spec=time period (e.g. 5 M=in last 5 minutes)); and
(spec)!$$(range)
: not recently departed from vicinity of (spec=time period (e.g. 5 M=in last 5 minutes)).

Values for "range" above can be any reasonable units such as 3K implies 3 Kilometers, 3M implies 3 Meters, 1 L implies 3 Miles, 3 F implies 3 Feet, etc. Values for "spec" above can be any reasonable time specification as described for TimeSpec (FIG. **30**B) and/or using qualifiers like "range" such as 3 W implies 3 Weeks, 3 D implies 3 Days, 3 H implies 3 Hours, 3M implies 3 Minutes, etc.

Resolving of conditions using atomic operators involves evaluating conditions (BNF grammar constructs) and additionally accessing similar data of LBX history **30** in some preferred embodiments. Atomic operator validation errors should result when inappropriately used.

Example syntactical embodiments of the "atomic profile match operator" atomic element include:

#: number of profile matches;
%: percentage of profile matches;
#(tag(s)): number of profile tag section matches (e.g. #(interests) compares one profile tag "interests"); and
% (tag(s)): percentage of profile tag section matches (e.g. #(interest,activities) compares a plurality of profile tags ("interests" and "activities").

In one embodiment of profiles maintained at MSs, a LBX singles/dating application maintains a MS profile for user's interests, tastes, likes, dislikes, etc. The ProfileMatch operators enable comparing user profiles under a variety of conditions, for example to cause an action of alerting a user that a person of interest is nearby. See FIGS. **77** and **78** for other profile information.

Atomic operators are context sensitive and take on their meaning in context to terms (i.e. BNF Grammar Term) they are used with. An alternate embodiment incorporates new appropriate atomic operators for use as CondOp operators, provided the result of the condition is a Boolean (e.g. term >=term results in a true or false). Also, while a syntactical form of parenthesis is not explicitly shown in the BNF grammar, the Conditions constructs explicitly defines how to make complex expressions with multiple conditions. Using parenthesis is one preferred syntactical embodiment for carrying out the Conditions construct. The intention of the BNF grammar is to end up with any reasonable conditional expression for evaluating to a Boolean True or False. Complex expression embodiments involving any conceivable operators, terms, order of evaluation (e.g. as syntactically represented with parentheses), and other arithmetic similarities, are certainly within the spirit and scope of this disclosure.

BNF grammar terms are to cover expressions containing conditions involving WDR fields (WDRTerm), situational locations, geofences (i.e. a geographic boundary identifying an area or space), two dimensional and three dimensional areas, two dimensional and three dimensional space, point in an area, point in space, movement amounts, movement distances, movement activity, MS IDs, MS group IDs, current mobile locations, past mobile locations, future mobile locations, nearness, distantness, newly near, newly afar, activities at locations (past, present, future), applications and context thereof in use at locations (past, present, future), etc. There are many various embodiments for specific supported operators used to provide interpretation to the terms. Certain operators, terms, and processing is presented for explanation and is in no way meant to limit the many other expression (BNF Grammar Expression) embodiments carrying the spirit of the disclosure.

The Command construct elaborates to atomic commands. The "atomic command" atomic element is a list of supported commands such as those found in the column headings of FIGS. **31**A through **31**E table (see discussions for FIGS. **31**A through **31**E). There are many commands, some popular commands being shown. The Operand construct elaborates to atomic operands. The "atomic operand" atomic element is a list of supported operands (data processing system objects) such as those found in the row headings of FIGS. **31**A through **31**E table (see discussions for FIGS. **31**A through **31**E). There are many operands, some popular operands being shown. For each command and operand combination, there may be anticipated parameters. The command and operand pair indicates how to interpret and process the parameters.

The constructs of Parameter, WDRTerm, AppTerm, Value and Data are appropriately interpreted within context of their usage. An optional time specification is made available when specifying charters (i.e. when charter is in effect), expressions (i.e. a plurality of conditions (e.g. with Conditions within Expressions construct)), a particular condition (e.g. with Condition elaborations within Condition construct), and actions (e.g. with Action elaborations within Action construct). One embodiment supports multiple Host specifica-

tions for a particular action. Some embodiments allow an Invocation to include invocations as parameters in a recursive manner so as to "bubble up" a resulting Boolean (e.g. fcn1(2, fcn2(p1, x, 45), 10) such that fcn2 may also have invocations for parameters. The conventional inside out evaluation order is implemented. Other embodiments support various types of invocations which contribute to the overall invocation result returned.

In alternate embodiments, an action can return a return code, for example to convey success, failure, or some other value(s) back to the point of performing the action. Such embodiments may support nesting of returned values in BNF grammar Parameters so as to affect the overall processing of actions. For example: action1(parameter(s), . . . , action2(. . . parameters . . .), . . . parameter(s)), and action2 may include returning value(s) from its parameters (which are actions).

Wildcarding is of value for broader specifications in a single specification. Wildcards may be used for BNF grammar specification wherever possible to broaden the scope of a particular specification (e.g. Condition, TimeSpec, etc).

FIGS. **31**A through **31**E depict a preferred embodiment set of command and operand candidates for Action Data Records (ADRs) (e.g. FIG. **37**B) facilitating the discussing of associated parameters (e.g. FIG. **37**C) of the ADRs of the present disclosure. Preferably, there are grammar specification privileges for governing every aspect of charters. Commands (atomic commands), operands (atomic operands), operators (atomic operators and Condop), parameters (Parameter), associated conditions (Condition and Condop), terms (Term), actions thereof (Action), associated data types thereof (Data), affected identities thereof (ID/IDType), and any other charter specification aspect, can be controlled by grammar specification privileges.

An "atomic command" is an enumeration shown in column headings (i.e. **101**, **103**, . . . etc) with an implied command meaning. FIG. **32**A shows what meaning is provided to some of the "atomic command" enumerations shown (also see FIG. **34**D). A plurality of commands can map to a single command meaning. This supports different words/phrases (e.g. spoken in a voice command interface) to produce the same resulting command so that different people specify commands with terminology, language, or (written) form they prefer. An "atomic operand" is an enumeration shown in row headings (i.e. **201**, **203**, . . . etc) with an implied operand meaning. FIG. **32**B shows what meaning is provided to some of the "atomic operand" enumerations shown (also see FIG. **34**D). A plurality of operands can map to a single operand meaning. This supports different words/phrases (e.g. spoken in a voice command interface) to produce the same resulting operand so that different people specify operands with terminology, language, or (written) form they prefer. Operands are also referred to as data processing system objects because they are common objects associated with data processing systems. FIGS. **31**A through **31**E demonstrate anticipated parameters for each combination of a command with an operand. There are potentially hundreds (or more) of commands and operands. This disclosure would be extremely large to cover all the different commands, operands, and parameters that may be reasonable. Only some examples with a small number of parameters are demonstrated in FIGS. **31**A through **31**E to facilitate discussions. There can be a large number of parameters for a command and operand pair. Each parameter, as shown by the BNF grammar, may be in many forms. In one preferred embodiment (not shown in BNF grammar), the Parameter construct of FIG. **30**E may also elaborate to a ParameterExpression which is any valid arithmetic expression that elaborates to one of the Parameter constructs (RHS) shown in the BNF Grammar. This allows specifying expressions which can be evaluated at run time for dynamically evaluating to a parameter for processing.

The combination of a command with an operand, and its set of associated parameters, form an action in the present disclosure, relative the BNF grammar discussed above. Some of the command/operand combinations overlap, or intersect, in functionality and/or parameters. In general, if parameters are not found (null specified) for an anticipated parameter position, a default is assumed (e.g. parameters of 5,,7 indicates three (3) parameters of 5, use default or ignore, and 7). Operands and parameters are preferably determined at executable code run time when referenced/accessed so that the underlying values may dynamically change as needed at executable code run time in the same references. For example, a variable set with constructs which elaborates to a command, operand, and parameters, can be instantiated in different contexts for completely different results. Also, a programming language enhanced with new syntax (e.g. as described in FIG. **51**) may include a loop for processing a single construct which causes completely different results at each loop iteration. The operand or parameter specification itself may be for a static value or dynamic value as determined by the reference used. An alternate embodiment elaborates values like a preprocessed macro ahead of time prior to processing for static command, operand, and parameter values. Combinations described by FIGS. **31**A through **31**E are discussed with flowcharts. In another embodiment, substitution (like parameter substitution discussed above for FIG. **30**A) can be used for replacing parameters at the time of invocation. In any case, Parameters can contain values which are static or dynamically changing up to the time of reference.

Parameters of atomic command processing will evaluate/resolve/elaborate to an appropriate data type and form for processing which is described by the #B matrices below (e.g. FIG. **63**B is the matrix for describing atomic send command processing). The #B descriptions provide the guide for the data types and forms supportable for the parameters. For example, an email body parameter may be a string, a file containing text, a variable which resolves to a string or file, etc. The BNF grammar is intended to be fully exploited in the many possible embodiments used for each parameter.

FIG. **32**A depicts a preferred embodiment of a National Language Support (NLS) directive command cross reference. Each "atomic command" has at least one associated directive, and in many cases a plurality of directives. Depending on an MS embodiment, a user may interact with the MS with typed text, voice control, selected graphical user interface text, symbols, or objects, or some other form of communication between the user and the MS. A directive (FIG. **32**A command and FIG. **32**B operand) embodies the MS recognized communication by the user. Directives can be a word, a phrase, a symbol, a set of symbols, something spoken, something displayed, or any other form of communications between a user and the MS. It is advantageous for a plurality of command directives mapped to an "atomic command" so a MS user is not limited with having to know the one command to operate the MS. The MS should cater to everyone with all anticipated user input from a diverse set of users which may be used to specify a command. This maximizes MS usability. The command directive is input to the MS for translating to the "atomic command". One preferred embodiment of a directive command cross reference **3202** maps a textual directive (Directive column) to a command ("atomic command" of Command column). In this embodiment, a user types a directive or speaks a directive to a voice control interface (ultimately converted to text). Cross reference **3204-1** demon-

strates an English language cross reference. Preferably, there is a cross reference for every language supported by the MS, for example, a Spanish cross reference **3204-2**, a Russian cross reference, a Chinese cross reference, and a cross reference for the L languages supported by the MS (i.e. **3204**-L being the final cross referenced language). Single byte character (e.g. Latin-1) and double byte character (e.g. Asian Pacific) encodings are supported. Commands disclosed are intended to be user friendly through support of native language, slang, or preferred command annunciation (e.g. in a voice control interface). FIG. **34**D enumerates some commands which may appear in a command cross reference **3202**.

FIG. **32**B depicts a preferred embodiment of a NLS directive operand cross reference. Each "atomic operand" has at least one associated directive, and in many cases a plurality of directives. It is advantageous for a plurality of operand directives mapped to an "atomic operand" so a MS user is not limited with having to know the one operand to operate the MS. The MS should cater to everyone with all anticipated user input from a diverse set of users which may be used to specify an operand. The directive is input to the MS for translating to the "atomic operand". One preferred embodiment of a directive operand cross reference **3252** maps a textual directive (Directive column) to an operand ("atomic operand" of Operand column). In this embodiment, a user types a directive or speaks a directive to a voice control interface (ultimately converted to text). Cross reference **3254-1** demonstrates an English language cross reference. Preferably, there is a cross reference for every language supported by the MS, for example, a Spanish cross reference **3254-2**, a Russian cross reference, a Chinese cross reference, and a cross reference for the L languages supported by the MS (i.e. **3254**-L being the final cross referenced language). Operands disclosed are intended to be user friendly through support of native language, slang, or preferred command annunciation (e.g. in a voice control interface). FIG. **34**D enumerates some operands which may appear in an operand cross reference **3252**.

In the preferred embodiment, Parameters are contextually determined upon the MS recognizing user directives, depending on the context in use at the time. In another embodiment, Parameters will also have directive mappings for being interpreted for MS processing, analogously to FIGS. **32**A and **32**B.

FIG. **33**A depicts a preferred embodiment American National Standards Institute (ANSI) X.409 encoding of the BNF grammar of FIGS. **30**A through **30**B for variables, variable instantiations and common grammar for BNF grammars of permissions and charters. A one superscript (1) is shown in constructs which may not be necessary in implementations since the next subordinate token can be parsed and deciphered on its own merit relative the overall length of the datastream containing the subordinate tokens. For example, a plural Variables construct and token is not necessary since an overall datastream length can be provided which contains sibling Variable constructs that can be parsed. Preferably, Variable assignments include the X.409 datastreams for the constructs or atomic elements as described in FIGS. **33**A through **33**C. FIG. **33**B depicts a preferred embodiment ANSI X.409 encoding of the BNF grammar of FIG. **30**C for permissions **10** and groups, and FIG. **33**C depicts a preferred embodiment ANSI X.409 encoding of the BNF grammar of FIGS. **30**D through **30**E for charters **12**. All of the X.409 encodings are preferably used to communicate information of permissions **10** and/or charters **12** (e.g. the BNF grammar constructs) between systems.

The preferred embodiment of a WDRTerm is a system well known WDR field/subfield variable name with two (2) leading underscore characters (e.g. source code references of: confidence refers to a confidence value of a WDR confidence field **1100***d*; _msyaw refers to a yaw value of a WDR location reference field **1100***f* MS yaw subfield). Some useful examples using a WDRTerm include:

- A specified MS, or group, WDR **1100** field (e.g. condition using field **1100***a* of (_I_msid !=George) & (_I_msid ^ChurchGroup));
- A specified MS, or group, WDR **1100** field or subfield value;
- A current MS, or group, WDR **1100** field (e.g. condition using field **1100***a* of (_msid !=George) & (_msid ^ ChurchGroup)); and
- A current MS, or group, WDR **1100** field or subfield value;

The preferred embodiment of an AppTerm is a system well known application variable name with a registered prefix, followed by an underscore character, followed by the variable name in context for the particular application (e.g. source code references of: M_source refers to a source email address of a received email for the registered MS email application which was registered with a "M" prefix; B_srchcriteria refers to the most recently specified search criteria used in the MS internet browser application which was registered with a "B" prefix). The preferred WDRTerm and AppTerm syntaxes provide user specifiable programmatic variable references for expressions/conditions to cause certain actions. The double underscore variable references refer to a WDR in process (e.g. inserted to queue **22** (_fldname), inbound to MS (_I_fldname), outbound from MS (_O_fldname)) at the particular MS. There is a system well known double underscore variable name for every field and subfield of a WDR as disclosed herein. The registered prefix name variable references always refer to data applicable to an object in process (e.g. specific data for: email just sent, email just received, phone call underway, phone call last made, phone call just received, calendar entry last posted, etc) within an application of the particular MS. There is a system well known underscore variable name for each exposed application data, and registering the prefix correlates the variable name to a particular MS application (see FIG. **53**).

An "atomic term" is another special type of user specifiable programmatic variable reference for expressions/conditions to cause certain actions. The preferred embodiment of an atomic term is a system well known variable name with a leading backslash (\) escape character (e.g. source code references of: \loc_my refers to the most recent MS location; \timestamp refers to the current MS system date/time in a date/time stamp format). There can be atomic terms to facilitate expression/condition specifications, some of which were described above.

FIGS. **33**A through **33**C demonstrate using the BNF grammar of FIGS. **30**A through **30**E to define an unambiguous datastream encoding which can be communicated between systems (e.g. MSs, or service and MS). Similarly, those skilled in the art recognize how to define a set of XML tags and relationships from the BNF grammar of FIGS. **30**A through **30**E for communicating an unambiguous XML datastream encoding which can be communicated between systems. For example, X.409 encoded tokens are translatable to XML tags that have scope between delimiters, and have attributes for those tags. The XML author may improve efficiency by making some constructs, which are subordinate to other constructs, into attributes (e.g. ID and IDType constructs as attributes to a Grantor and/or Grantee XML tag). The XML author may also decide to have some XML tags self

contained (e.g. <History creatordt=" . . . " creatorid=" . . . "/> or provide nesting, for example to accommodate an unpredictable plurality of subordinate items (e.g. <Permission . . . > . . . <Grantor userid="joe"/> . . . <Grantee groupid="dept1"/> . . . <Grantee groupid="dept43"/> . . . <Grantee groupid="dept9870"/> . . . </Permission>). It is a straightforward matter for translating the BNF grammar of FIGS. **30**A through **30**E into an efficiently processed XML encoding for communications between MSs. An appropriate XML header will identify the datastream (and version) to the receiving system (like HTML, WML, etc) and the receiving system (e.g. MS) will process accordingly using the present disclosure guide for proper parsing to internalize to a suitable processable format (e.g. FIGS. **34**A through **34**G, FIGS. **35**A through **37**C, FIG. **52**, or another suitable format per disclosure). See FIG. **54** for one example of an XML encoding.

FIGS. **34**A through **34**G depict preferred embodiment C programming source code header file contents, derived from the grammar of FIGS. **30**A through **30**E. A C example was selected so that the embodiment was purely data in nature. Another preferred embodiment utilizes an object oriented programming source code (e.g. C++, C#, or Java), but those examples mix data and object code in defining relationships. A preferred object oriented architecture would create objects for BNF grammar constructs that contain applicable processing data and code. The object hierarchy would then equate to construct relationships. Nevertheless, a purely data form of source code is demonstrated by FIGS. **34**A through **34**G (and FIG. **52**) to facilitate understanding. Those skilled in the relevant arts know how to embody the BNF grammar of FIG. **30**A through **30**E in a particular programming source code. The C programming source code may be used for:

- Parsing, processing, and/or internalizing a derivative X.409 encoding of the BNF grammar of FIGS. **30**A through **30**E (e.g. FIGS. **33**A through **33**C);
- Parsing, processing, and/or internalizing a derivative XML encoding of the BNF grammar of FIGS. **30**A through **30**E;
- Compiler parsing, processing, and/or internalizing of a programming language processing form of the BNF grammar of FIGS. **30**A through **30**E;
- Interpreter parsing, processing, and/or internalizing of a programming language processing form of the BNF grammar of FIGS. **30**A through **30**E;
- Internalized representation of permissions **10**, groups (data **8**) and/or charters **12** to data processing system memory;
- Internalized representation of permissions **10**, groups (data **8**) and/or charters **12** to data processing system storage; and/or
- Parsing, processing, and/or internalizing any particular derivative form, or subset, of the BNF grammar of FIGS. **30**A through **30**E.

Source code header information is well understood by those skilled in the relevant art in light of the BNF grammar disclosed. The example does make certain assumptions which are easily altered depending on specificities of a derivative form, or subset, of the grammar of FIGS. **30**A through **30**E. Assumptions are easily modified for "good" implementations through modification of isolated constants in the header file:

- TLV tokens are assumed to occupy 2 bytes in length;
- TLV length bytes are assumed to occupy 4 bytes in length;
- Some of the header definitions may be used solely for processing X.409 encodings in which case they can be removed depending on the context of source code use;
- Data structure linkage;
- Data structure form without affecting objective semantics;
- Data structure field definitions;
- Unsigned character type is used for data that can be a typecast byte stream, and pointers to unsigned character is used for pointers to data that can be typecast;
- Source code syntax; or
- Other aspects of the source code which are adaptable to a particular derivative form, or subset, of the BNF grammar of FIGS. **30**A through **30**E.

The TIMESPEC structure of FIG. **34**E preferably utilizes a well performing Julian date/time format. Julian date/time formats allows using unambiguous floating point numbers for date/time stamps. This provides maximum performance for storage, database queries, and data manipulation. Open ended periods of time use an unspecified start, or end data/time stamp, as appropriate (i.e. DT_NOENDSPEC or DT_NOSTARTSPEC). A known implemented minimal time granulation used in Julian date/time stamps can be decrement or incremented by one (1) as appropriate to provide a non-inclusive date/time stamp period delimiter in a range specification (e.g. > date/time stamp).

The VAR structure provides a pointer to a datastream which can be typecast (if applicable in embodiments which elaborate the variable prior to being instantiated, or referenced), or later processed. Variables are preferably not elaborated/evaluated until instantiated or referenced. For example, the variable assigned value(s) which are parsed from an encoding remains unprocessed (e.g. stays in X.409 datastream encoded form) until instantiated. Enough space is dynamically allocated for the value(s) (e.g. per length of variable's value(s)) (e.g. X.409 encoding form), the variable's value (e.g. X.409 encoding) is copied to the allocated space, and the v.value pointer is set to the start of the allocated space. The v.value pointer will be used later when the variable is instantiated (to then parse and process the variable value(s) when at the context they are instantiated).

An alternate embodiment to the PERMISSION structure of FIG. **34**F may not require the grantor fields (e.g. grantor, gortype) since the data processing system owning the data may only maintain permissions for the grantor (e.g. the MS user). An alternate embodiment to the CHARTER structure of FIG. **34**G may not require the grantee fields (e.g. grantee, geetype) or the grantor fields (e.g. grantor, gortype) since the data processing system owning the data may only maintain charters for that user at his MS. Another embodiment to the CHARTER structure of FIG. **34**G may not require the grantor fields (e.g. grantor, gortype) since the data processing system owning the data may be self explanatory for the Grantor identity (e.g. charters used at MS of Grantor).

FIGS. **35**A through **37**C, and FIG. **53**, illustrate data records, for example maintained in an SQL database, or maintained in record form by a data processing system. Depending on the embodiment, some data record fields disclosed may be multi-part fields (i.e. have sub-fields), fixed length records, varying length records, or a combination with field(s) in one form or another. Some data record field embodiments will use anticipated fixed length record positions for subfields that can contain useful data, or a null value (e.g. −1). Other embodiments may use varying length fields depending on the number of sub-fields to be populated, or may use varying length fields and/or sub-fields which have tags indicating their presence. Other embodiments will define additional data record fields to prevent putting more than one accessible data item in one field. In any case, processing will have means for knowing whether a value is present or not, and for which field (or sub-field) it is present. Absence in data may be indicated with a null indicator (−1), or indicated with its lack of being there (e.g. varying length record embodiments). Fields described

may be converted: a) prior to storing; or b) after accessing; or c) by storage interface processing; for standardized processing. Fields described may not be converted (i.e. used as is).

FIG. **35**A depicts a preferred embodiment of a Granting Data Record (GDR) **3500** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. A GDR **3500** is the main data record for defining a granting of permissions **10**, or charters **12**. A granting identifier (granting ID) field **3500***a* contains a unique number generated for the record **3500** to distinguish it from all other records **3500** maintained. For example, in a Microsoft SQL Server deployment, granting ID field **3500***a* is a primary key column. Another embodiment uses the correlation generation techniques described above to ensure a unique number is generated. Field **3500***a* facilitates well performing searches, updates, deletes, and other I/O (input/output) interfaces. Field **3500***a* may match (for joining) a field **3520***b* or **3700***a*, depending on the GDR type (GDR type field **3500***t* with value of Permission or Charter). A granting type field **3500***t* distinguishes the type of GDR (Permission or Charter) for: a Grantor granting all privileges to a Grantee (i.e. Permission (e.g. ID field **3500***a* unique across GDRs but not used to join other data records)), a Grantor granting specific privilege(s) and/or grants of privileges (permission(s)) to a Grantee ((i.e. Permission (e.g. ascendant ID field **3520***b* value in ID field **3500***a*)), and a Grantor granting enablement of a charter to a Grantee ((i.e. Charter (e.g. charter ID field **3700***a* value in ID field **3500***a*)). An owner information (info) field **3500***b* provides who the owner (creator and/or maintainer) is of the GDR **3500**. Depending on embodiments, or how the GDR **3500** was created, owner info field **3500***b* may contain data like the ID and type pair as defined for fields **3500***c* and **3500***d*, or fields **3500***e* and **3500***f*. An alternate embodiment to owner info field **3500***b* is two (2) fields: owner info ID field **3500***b*–1 and owner info type field **3500***b*–2. Yet another embodiment removes field **3500***b* because MS user (e.g. the grantor) information is understood to be the owner of the GDR **3500**. The owner field **3500***b* may become important in user impersonation. A grantor ID field **3500***c* provides an identifier of the granting grantor and a grantor type field **3500***d* provides the type of the grantor ID field **3500***c*. A grantee ID field **3500***e* provides an identifier of the granting grantee and a grantee type field **3500***f* provides the type of the grantee ID field **3500***e*.

FIG. **35**B depicts a preferred embodiment of a Grant Data Record (GRTDR) **3510** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. A GRTDR **3510** is the main data record for defining a grant. A grant identifier (grant ID) field **3510***a* contains a unique number generated for the record **3510** to distinguish it from all other records **3510** maintained. Field **3510***a* is to be maintained similarly to as described for field **3500***a* (e.g. primary key column, correlation generation, facilitates well performing I/O). An owner information (info) field **3510***b* provides who the owner (creator and/or maintainer) is of the GRTDR **3510**. Field **3510***b* is to be maintained similarly to as described for field **3500***b* (e.g. embodiments for like ID and type pair, two (2) fields, removal because MS user information understood to be owner). A grant name field **3510***c* provides the name of the grant.

FIG. **35**C depicts a preferred embodiment of a Generic Assignment Data Record (GADR) **3520** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. A GADR **3520** is the main data record for defining an assignment relationship between data records. The assignment relationship can be viewed as a container relationship, or a parent-child relationship such as in a tree structure. An ascendant type field **3520***a* contains the type of parent (or container) data record in the relationship. Values maintained to field **3520***a* include Permission, Grant, or Group. An ascendant ID field **3520***b* provides an identifier of the parent (or container) data record in the relationship (used for joining data records in queries in an SQL embodiment). Values maintained to field **3520***b* include values of granting ID field **3500***a*, grant ID field **3510***a*, or group ID field **3540***a*. A descendant type field **3520***c* contains the type of child (or contained) data record in the relationship. Values maintained to field **3520***c* include Grant, Privilege, Group, or ID Type (e.g. Grantor or Grantee ID type). A descendant ID field **3520***d* provides an identifier of the child (or contained) data record in the relationship (used in joining data records in queries in an SQL embodiment). Values maintained to field **3520***d* include values of grant ID field **3510***a*, privilege identifier (i.e. "atomic privilege for assignment"), group ID field **3540***a*, ID field **3500***c*, or ID field **3500***e*. Records **3520** (key for list below is descendant first; ascendant last (i.e. " . . . in a . . . ")) are used to represent:

Grant(s) (the descendants) in a permission (the ascendant);
Privilege(s) in a permission;
Grant(s) in a grant (e.g. tree structure of grant names);
Privilege(s) in a grant;
Groups(s) in a group (e.g. tree structure of group names);
IDs in a group (e.g. group of grantors and/or grantees); and/or
Other parent/child relationships of data records disclosed.

An alternate embodiment will define distinct record definitions (e.g. **3520**-*z*) for any subset of relationships described to prevent data access performance of one relationship from impacting performance accesses of another relationship maintained. For example, in an SQL embodiment, there may be two (2) tables: one for handling three (3) of the relationships described, and another for handling all other relationships described. In another SQL example, six (6) distinct tables could be defined when there are only six (6) relationships to maintain. Each of the distinct tables could have only two (2) fields defined for the relationship (i.e. ascendant ID and descendant ID). The type fields may not be required since it would be known that each table handles a single type of relationship (i.e. GADR-grant-to-permission, GADR-privilege-to-permission, GADR-grant-to-grant, GADR-privilege-to-grant, GADR-group-to-group and GADR-ID-to-group). Performance considerations may provide good reason to separate out relationships maintained to distinct tables (or records).

FIG. **35**D depicts a preferred embodiment of a Privilege Data Record (PDR) **3530** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. A privilege ID field **3530***a* contains a unique number associated to a supported privilege (i.e. "atomic privilege for assignment"). Field **3530***a* associates a MS relevance field **3530***b* to a particular privilege for indicating the MS types which apply to a privilege. There should not be more than one PDR **3530** at a MS with matching fields **3530***a* since the associated field **3530***b* defines the MS types which are relevant for that privilege. If there is no record **3530** for a particular privilege, then it is preferably assumed that all MSs participate with the privilege. MS relevance field **3530***b* is preferably a bit mask accommodating all anticipated MS types, such that a 1 in a predefined MS type bit position indicates the MS participates with the privilege, and a 0 in a predefined MS type bit position indicates the MS does not participate with the privilege. Optimally, there are no records **3530** at a MS which implies all supported privileges interoperate fully with other MSs according to the present disclosure.

FIG. **35**E depicts a preferred embodiment of a Group Data Record (GRPDR) **3540** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. A GRPDR **3540** is the main data record for defining a group. A group identifier (group ID) field **3540***a* contains a unique number generated for the record **3540** to distinguish it from all other records **3540** maintained. Field **3540***a* is to be maintained similarly to as described for field **3500***a* (e.g. primary key column, correlation generation, facilitates well performing I/O). An owner information (info) field **3540***b* provides who the owner (creator and/or maintainer) is of the GRPDR **3540**. Field **3540***b* is to be maintained similarly to as described for field **3500***b* (e.g. embodiments for like ID and type pair, two (2) fields, removal because MS user information understood to be owner). A group name field **3540***c* provides the name of the group.

FIG. **36**A depicts a preferred embodiment of a Description Data Record (DDR) **3600** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. A DDR **3600** is for maintaining description information for certain constructs. A description ID field **3600***a* provides an identifier of the data record associated to the description field **3600***c*. For example, values maintained to field **3600***a* are used for joining data records in queries in an SQL embodiment. Values maintained to field **3600***a* include values of granting ID field **3500***a*, grant ID field **3510***a*, a privilege ID (e.g. as candidate to field **3530***a*), ID field **3500***c*, ID field **3500***e*, charter ID field **3700***a*, action ID field **3750***a*, parameter ID field **3775***a*, group ID field **3540***a*, or any other ID field for associating a description. A description type field **3600***b* contains the type of data record to be associated (e.g. joined) to the description field **3600***c*. Values maintained to field **3600***b* include Permission, Grant, Privilege, ID, Charter, Action, Parameter, or Group in accordance with a value of field **3600***a*. Field **3600***c* contains a description, for example a user defined text string, to be associated to the data described by fields **3600***a* and **3600***b*. Alternate embodiments will move the description data to a new field of the data record being associated to, or distinct record definitions **3600**-*y* may be defined for any subset of relationship/association to prevent data access performance of one relationship/association from impacting performance accesses of another relationship/association maintained (analogous to distinct embodiments for GADR **3520**).

FIG. **36**B depicts a preferred embodiment of a History Data Record (HDR) **3620** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. A HDR **3620** is for maintaining history information for certain constructs. A history ID field **3620***a* provides an identifier of the data record associated to the history field **3620***c*. For example, values maintained to field **3620***a* are used for joining data records in queries in an SQL embodiment. Values maintained to field **3620***a* include values of granting ID field **3500***a*, grant ID field **3510***a*, a privilege ID (e.g. as candidate to field **3530***a*), ID field **3500***c*, ID field **3500***e*, charter ID field **3700***a*, action ID field **3750***a*, parameter ID field **3775***a*, group ID field **3540***a*, or any other ID field for associating a history. A history type field **3620***b* contains the type of data record to be associated (e.g. joined) to the history field **3620***c*. Values maintained to field **3620***b* include Permission, Grant, Privilege, ID, Charter, Action, Parameter, or Group in accordance with a value of field **3620***a*. Field **3620***c* contains a history, for example a collection of fields for describing the creation and/or maintenance of data associated to the data described by fields **3620***a* and **3620***b*. Alternate embodiments will move the history data to new field(s) of the data record being associated to, or distinct record definitions **3620**-*x* may be defined for any subset of relationship/association to prevent data access performance of one relationship/association from impacting performance accesses of another relationship/association maintained (analogous to distinct embodiments for GADR **3520**). Another embodiment may break out subfields of field **3620***c* to fields **3620***c*−1, **3620***c*−2, **3620***c*−3, etc. for individual fields accesses (e.g. see CreatorInfo and ModifierInfo sub-fields).

FIG. **36**C depicts a preferred embodiment of a Time specification Data Record (TDR) **3640** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. A TDR **3640** is for maintaining time spec information for certain constructs. A time spec ID field **3640***a* provides an identifier of the data record associated to the time spec field **3640***c*. For example, values maintained to field **3640***a* are used for joining data records in queries in an SQL embodiment. Values maintained to field **3640***a* include values of granting ID field **3500***a*, grant ID field **3510***a*, a privilege ID (e.g. as candidate to field **3530***a*), charter ID field **3700***a*, action ID field **3750***a*, or any other ID field for associating a time spec (specification). A time spec type field **3640***b* contains the type of data record to be associated (e.g. joined) to the time spec field **3640***c*. Values maintained to field **3640***b* include Permission, Grant, Privilege, Charter, or Action in accordance with a value of field **3640***a*. Field **3640***c* contains a time spec, for example one or more fields for describing the date/time(s) for which the data associated to the data described by fields **3640***a* and **3640***b* is applicable, enabled, or active. For example, permissions can be granted as enabled for particular time period(s). Alternate embodiments will move the time spec data to new field(s) of the data record being associated to, or distinct record definitions **3640**-*w* may be defined for any subset of relationship/association to prevent data access performance of one relationship/association from impacting performance accesses of another relationship/association maintained (analogous to distinct embodiments for GADR **3520**). Another embodiment may break out subfields of field **3640***c* to fields **3640***c*−1, **3640***c*−2, **3620***c*−3, etc. Field **3640***c* (and sub-fields if embodiment applicable) can describe specific date/time(s) or date/time period(s). Yet another embodiment, maintains plural TDRs for a data record of ID field **3640***a*. Field **3640***c* is intended to qualify the associated data of fields **3640***a* and **3640***b* for being applicable, enabled, or active at future time(s), past time(s), or current time(s). An alternate embodiment of field **3640***c* may include a special tense qualifier as defined below:

- Past ("P"): indicates that the associated data record (e.g. permission, charter, action, etc) applies to all WDR information maintained to LBX History **30**;
- Self Past ("SP"): indicates that the associated data record (e.g. permission, charter, action, etc) applies to only WDR information maintained to LBX History **30** for the MS owning history **30**;
- Other Past ("OP"): indicates that the associated data record (e.g. permission, charter, action, etc) applies to only WDR information maintained to LBX History **30** for all MSs other than the one owning history **30**;
- Future ("F"): indicates that the associated data record (e.g. permission, charter, action, etc) applies to all WDRs created/received (e.g. inserted to queue **22**) in the future by the MS (i.e. after configuration made);
- Self Future ("SF"): indicates that the associated data record (e.g. permission, charter, action, etc) applies to all WDRs created in the future (e.g. inserted to queue **22**) by the MS for its own whereabouts (i.e. after configuration made);

Other Future ("OF"): indicates that the associated data record (e.g. permission, charter, action, etc) applies to all WDRs received (e.g. inserted to queue **22**) in the future by the MS for other MS whereabouts (i.e. after configuration made);

All ("A"): indicates that the associated data record (e.g. permission, charter, action, etc) applies to all WDRs created/received in the future by the MS (i.e. after configuration made) and WDRs already contained by queue **22**;

Self All ("SA"): indicates that the associated data record (e.g. permission, charter, action, etc) applies to all WDRs created in the future by the MS for its own whereabouts (i.e. after configuration made) and WDRs already contained by queue **22** for the MS;

Other All ("OA"): indicates that the associated data record (e.g. permission, charter, action, etc) applies to all WDRs received in the future by the MS for other MS whereabouts (i.e. after configuration made) and WDRs already contained by queue **22** for other MSs; and/or

Any combination of above (e.g. "SF,OA,OP")

A syntactical equivalent may be specified for subsequent internalization causing configurations to immediately take effect. Another embodiment qualifies which set of MSs to apply time specification for, but this is already accomplished below in the preferred embodiment through specifications of conditions. Yet another embodiment provides an additional qualifier specification for which WDRs to apply the time specification: WDRs maintained by the MS (e.g., to queue **22**), inbound WDRs as communicated to the MS, outbound WDRs as communicated from the MS; for enabling applying of time specifications before and/or after privileges/charters are applied to WDRs with respect to an MS. Blocks **3970**, **4670** and **4470** may be amended to include processing for immediately checking historical information maintained at the MS which privileges/charters have relevance, for example after specifying a historical time specification or special tense qualifier.

FIG. **36**D depicts a preferred embodiment of a Variable Data Record (VDR) **3660** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. A VDR **3660** contains variable information that may be instantiated. A record **3660** provide a single place to define an encoding that is instantiated in many places. One advantage is for saving on encoding sizes. An owner information (info) field **3660***a* provides who the owner (creator and/or maintainer) is of the VDR **3660**. Field **3660***a* is to be maintained similarly to as described for field **3500***b* (e.g. embodiments for like ID and type pair, two (2) fields, removal because grantor information understood to be owner). Variable name field **3660***b* contains the variable name string, variable type field **3660***c* contains the variable type, and variable value field **3660***d* contains the value(s) of the variable for instantiation. Preferably, field **3660***d* remains in its original form until the variable is instantiated. For example, in an X.409 embodiment, field **3660***d* contains the X.409 encoding datastream (including the overall length for starting bytes) of the variable value. In a programming source, XML, or other syntactical embodiment (of grammar of FIGS. **30**A through **30**F), field **3660***d* contains the unelaborated syntax in text form for later processing (e.g. stack processing). Thus, field **3660***d* may be a BLOB (Binary Large Object) or text. Preferably, field **3660***d* is not elaborated, or internalized, until instantiated. When a variable is set to another variable name, field **3660***d* preferably contains the variable name and the variable type field **3660***c* indicates Variable. Preferably, field **3660***d* handles varying length data well for performance, or an alternate embodiment will provide additional VDR field(s) to facilitate performance.

FIG. **37**A depicts a preferred embodiment of a Charter Data Record (CDR) **3700** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. A CDR **3700** is the main data record for defining a charter. A charter identifier (charter ID) field **3700***a* contains a unique number generated for the record **3700** to distinguish it from all other records **3700** maintained. Field **3700***a* is to be maintained similarly to as described for field **3500***a* (e.g. primary key column, correlation generation, facilitates well performing I/O). Grantee and Grantor information is linked to with a match of field **3700***a* with **3500***a*. An alternate embodiment will require no Grantee or Grantor specification for a charter (e.g. charters maintained and used at the user's MS). An owner information (info) field **3700***b* provides who the owner (creator and/or maintainer) is of the CDR **3700**. Field **3700***b* is to be maintained similarly to as described for field **3500***b* (e.g. embodiments for like ID and type pair, two (2) fields, removal because MS user information understood to be owner). An expression field **3700***c* contains the expression containing one or more conditions for when to perform action(s) of action field **3700***d*. Preferably, field **3700***c* remains in its original form until the conditions are to be elaborated, processed, or internalized. For example, in one X.409 embodiment, field **3700***c* contains the X.409 encoding datastream for the entire Expression TLV. In the preferred syntactical embodiment (programming source code, XML encoding, programming source code enhancement, or the like), field **3700***c* contains the unelaborated syntax in text form for later stack processing of conditions and terms and their subordinate constructs. Thus, field **3700***c* may be a BLOB (Binary Large Object) or (preferably) text. An alternate embodiment to field **3700***c* may use General Assignment Data Records (GADRs) **3520** to assign condition identifier fields of a new condition data record to charter identifier fields **3700***a* (to prevent a single field from holding an unpredictable number of conditions for the charter of record **3700**). Actions field **3700***d* contains an ordered list of one or more action identifiers **3750***a* of actions to be performed when the expression of field **3700***c* is evaluated to TRUE. For example, in the preferred syntactical embodiment, when actions field **3700***d* contains "45,2356,9738", the action identifier fields **3750***a* have been identified as an ordered list of actions **45**, **2356** and **9738** which are each an action identifier contained in an ADR **3750** field **3750***a*. An alternate embodiment to field **3700***d* will use General Assignment Data Records (GADRs) **3520** to assign action identifier fields **3750***a* to charter identifier fields **3700***a* (to prevent a single field from holding an unpredictable number of actions for the charter of record **3700**). Another alternative embodiment may include Grantor and Grantee information as part of the CDR (e.g. new fields **3700***e* through **3700***h* like fields **3500***c* through **3500***f*.

FIG. **37**B depicts a preferred embodiment of an Action Data Record (ADR) **3750** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. An action identifier (action ID) field **3750***a* contains a unique number generated for the record **3750** to distinguish it from all other records **3750** maintained. Field **3750***a* is to be maintained similarly to as described for field **3500***a* (e.g. primary key column, correlation generation, facilitates well performing I/O). An owner information (info) field **3750***b* provides who the owner (creator and/or maintainer) is of the ADR **3750**. Field **3750***b* is to be maintained similarly to as described for field **3500***b* (e.g. embodiments for like ID and type pair, two (2) fields, removal because MS

user information understood to be owner). Host field **3750***c* contains the host (if not null) for where the action is to take place. An alternate embodiment allows multiple host specification(s) for the action. Host type field **3750***d* qualifies the host field **3750***c* for the type of host(s) to perform the action (helps interpret field **3750***c*). An alternate embodiment allows multiple host type specifications for multiple host specifications for the action. Yet another embodiment uses a single host field **3750***c* to join to a new table for gathering all applicable hosts for the action. Command field **3750***e* contains an "atomic command" (such as those found at the top of FIG. **34**D), operand field **3750***f* contains an "atomic operand" (e.g. such as those found at the bottom of FIG. **34**D), and parameter IDs field **3750***g* contains a list of null, one or more parameter identifiers **3775***a* (an ordered list) for parameters in accordance with the combination of command field **3750***e* and operand field **3750***f* (see FIGS. **31**A through **31**E for example parameters). There is a list of supported commands, list of supported operands, and a set of appropriate parameters depending on the combination of a particular command with a particular operand. In the preferred syntactical embodiment, when parameter IDs field **3750***g* contains "234,18790", the parameter IDs fields **3775***a* have been identified as an ordered list of parameters **234** and **18790** which are each a parameter identifier contained in a record **3775** field **3775***a*. An alternate embodiment to field **3750***g* will use General Assignment Data Records (GADRs) **3520** to assign parameter identifier fields **3775***a* to action identifier fields **3750***a* (to prevent a single field from holding an unpredictable number of parameters for the action of record **3750**).

FIG. **37**C depicts a preferred embodiment of a Parameter Data Record (PARMDR) **3775** for discussing operations of the present disclosure, derived from the grammar of FIGS. **30**A through **30**E. A parameter identifier (parameter ID) field **3775***a* contains a unique number generated for the record **3775** to distinguish it from all other records **3775** maintained. Field **3775***a* is to be maintained similarly to as described for field **3500***a* (e.g. primary key column, correlation generation, facilitates well performing I/O). An owner information (info) field **3775***b* provides who the owner (creator and/or maintainer) is of the record **3775**. Field **3775***b* is to be maintained similarly to as described for field **3500***b* (e.g. embodiments for like ID and type pair, two (2) fields, removal because MS user information understood to be owner). Parameters field **3775***c* contains one or more parameters pointed to by data of field **3750***g*, preferably in a conveniently parsed form. Field **3750***g* can point to a single record **3775** which contains a plurality of parameters in field **3775***c*, or field **3750***g* can specify a plurality of parameters pointing to plural records **3775**, each containing parameter information in fields **3775***c*.

In one embodiment, data can be maintained to data records of FIGS. **35**A through **37**C, and FIG. **53**, such that it is marked as enabled or disabled (e.g. additional column in SQL table for enabled/disabled). In another embodiment, a record is configured in disabled form and then subsequently enabled, for example with a user interface. Any subset of data records may be enabled or disabled as a related set. Privileges may be configured for which subsets can be enabled or disabled by a user. In another embodiment, privileges themselves enable or disable a data record, a subset of data records, a subset of data record types, or a subset of data of data records.

Data records were derived from the BNF grammar of FIGS. **30**A through **30**E. Other data record embodiments may exist. In a preferred embodiment, data records of FIGS. **35**A through **37**C are maintained to persistent storage of the MS. A MS used for the first time should be loaded with a default set of data (e.g. starter templates containing defaulted data) preloaded to the data records for user convenience. Loading may occur from local storage or from remotely loading, for example over a communications channel when first initializing the MS (e.g. enhanced block **1214** for additionally ensuring the data records are initialized, in particular for the first startup of an MS). Owner fields (e.g. field **3500***b*) for preloaded data are preferably set to a system identity for access and use by all users. Preferably, a user cannot delete any of the system preloaded data. While the data records themselves are enough to operate permissions **10** and charters **12** at the MS after startup, a better performing internalization may be preferred. For example, block **1216** can be enhanced for additionally using data records to internalize to a non-persistent well performing form such as compiled C encoding of FIGS. **34**A through **34**G (also see FIG. **52**), and block **2822** can be enhanced for additionally using the internalized data to write out to data records maintained in persistent storage. Any compiled/interpreted programming source code may be used without departing from the spirit and scope of the disclosure. FIGS. **34**A through **34**G (also see FIG. **52**) are an example, but may provide an internalized form for processing. In any case, many examples are provided for encoding permissions **10** and charters **12**. Continuing with the data record examples, for example a persistent storage form of data records in a MS local SQL database (e.g. a data record corresponds to a particular SQL table, and data record fields correspond to the SQL table columns), flowcharts **38** through **48**B are provided for configuration of permissions **10** and charters **12**. Data records are to be maintained in a suitable MS performance conscious form (may not be an SQL database). An "s" is added as a suffix to disclosed acronyms (e.g. GDR) to reference a plural version of the acronym (e.g. GDRs=Granting Data Records).

FIGS. **35**A through **37**C assume an unlimited number of records (e.g. objects) to accomplish a plurality of objects (e.g. BNF grammar constructs). In various embodiments, a high maximum number plurality of the BNF grammar derived objects is supported wherever possible. In various embodiments, any MS storage or memory means, local or remotely attached, can be used for storing information of an implemented derivative of the BNF grammar of this disclosure. Also, various embodiments may use a different model or schema to carry out functionality disclosed. Various embodiments may use an SQL database (e.g. Oracle, SQL Server, Informix, DB2, etc. for storing information, or a non-SQL database form (e.g. data or record retrieval system, or any interface for accessible record formatted data).

It is anticipated that management of permissions **10** and charters **12** be as simple and as lean as possible on an MS. Therefore, a reasonably small subset of the FIGS. **30**A through **30**E grammar is preferably implemented. While FIGS. **35**A through **48**B demonstrate a significantly large derivative of the BNF grammar, the reader should appreciate that this is to "cover all bases" of consideration, and is not necessarily a derivative to be incorporated on a MS of limited processing capability and resources. A preferred embodiment is discussed, but much smaller derivatives are even more preferred on many MSs. Appropriate semaphore lock windows are assumed incorporated when multiple asynchronous threads can access the same data concurrently.

FIG. **38** depicts a flowchart for describing a preferred embodiment of MS permissions configuration processing of block **1478**. FIG. **38** is of Self Management Processing code **18**. Processing starts at block **3802** and continues to block **3804** where a list of permissions configuration options are presented to the user. Thereafter, block **3806** waits for a user action in response to options presented. Block **3806** continues

to block **3808** when a user action has been detected. If block **3808** determines the user selected to configure permissions data, then the user configures permissions data at block **3810** (see FIG. **39**A) and processing continues back to block **3804**. If block **3808** determines the user did not select to configure permissions data, then processing continues to block **3812**. If block **3812** determines the user selected to configure grants data, then the user configures grants data at block **3814** (see FIG. **40**A) and processing continues back to block **3804**. If block **3812** determines the user did not select to configure grants data, then processing continues to block **3816**. If block **3816** determines the user selected to configure groups data, then the user configures groups data at block **3818** (see FIG. **41**A) and processing continues back to block **3804**. If block **3816** determines the user did not select to configure groups data, then processing continues to block **3820**. If block **3820** determines the user selected to view other's groups data, then block **3822** invokes the view other's info processing of FIG. **42** with GROUP_INFO as a parameter (for viewing other's groups data information) and processing continues back to block **3804**. If block **3820** determines the user did not select to view other's groups data, then processing continues to block **3824**. If block **3824** determines the user selected to view other's permissions data, then block **3826** invokes the view other's info processing of FIG. **42** with PERMISSION_INFO as a parameter (for viewing other's permissions data information) and processing continues back to block **3804**. If block **3824** determines the user did not select to view other's permissions data, then processing continues to block **3828**. If block **3828** determines the user selected to view other's grants data, then block **3830** invokes the view other's info processing of FIG. **42** with GRANT_INFO as a parameter (for viewing other's grants data information) and processing continues back to block **3804**. If block **3828** determines the user did not select to view other's grants data, then processing continues to block **3832**. If block **3832** determines the user selected to send permissions data, then block **3834** invokes the send data processing of FIG. **44**A with PERMISSION_INFO as a parameter (for sending permissions data) and processing continues back to block **3804**. If block **3832** determines the user did not select to send permissions data, then processing continues to block **3836**. If block **3836** determines the user selected to configure accepting permissions, then block **3838** invokes the configure acceptance processing of FIG. **43** with PERMISSION_INFO as a parameter (for configuring acceptance of permissions data) and processing continues back to block **3804**. If block **3836** determines the user did not select to configure accepting permissions, then processing continues to block **3840**. If block **3840** determines the user selected to exit block **1478** processing, then block **3842** completes block **1478** processing. If block **3840** determines the user did not select to exit, then processing continues to block **3844** where all other user actions detected at block **3806** are appropriately handled, and processing continues back to block **3804**.

In an alternate embodiment where the MS maintains GDRs **3500**, GRTDRs **3510**, GADRs **3520**, PDRs **3530** and GRPDRs **3540** (and their associated data records DDRs, HDRs and TDRs) at the MS where they were configured, FIG. **38** may not provide blocks **3820** through **3830**. The MS may be aware of its user permissions and need not share the data (i.e. self contained). In some embodiments, options **3820** through **3830** cause access to locally maintained data for others (other users, MSs, etc) or cause remote access to data when needed (e.g. from the remote MSs). In the embodiment where no data is maintained locally for others, blocks **3832** through **3838** may not be necessary. The preferred embodiment is to locally maintain permissions data for the MS user and others (e.g. MS users) which are relevant to provide the richest set of permissions governing MS processing at the MS.

FIGS. **39**A through **39**B depict flowcharts for describing a preferred embodiment of MS user interface processing for permissions configuration of block **3810**. With reference now to FIG. **39**A, processing starts at block **3902**, continues to block **3904** for initialization (e.g. a start using database command), and then to block **3906** where groups the user is a member of are accessed. Block **3906** retrieves all GRPDRs **3540** joined to GADRs **3520** such that the descendant type field **3520***c* and descendant ID field **3520***d* match the user information, and the ascendant type field **3520***a* is set to Group and the ascendant ID field **3520***b* matches the group ID field **3540***a*. While there may be different types of groups as defined for the BNF grammar, the GRPDR is a derivative embodiment which happens to not distinguish. Alternate embodiments may carry a group type field to select appropriate records by group type. Yet another embodiment may not have a block **3906** with processing at block **3908** for gathering data additionally by groups the user is a member of. Block **3906** continues to block **3908**.

Block **3908** accesses all GDRs (e.g. all rows from a GDR SQL table) for the user of FIG. **39**A matching field **3500***t* to Permission, and the owner information of the GDRs (e.g. user information matches field **3500***b*) to the user and to groups the user is a member of (e.g. group information matches field **3500***b* (e.g. owner type=group, owner id=a group ID field **3540***a* from block **3906**). The GDRs are additionally joined (e.g. SQL join) with DDRs and TDRs (e.g. fields **3600***b* and **3640***b*=Permission and by matching ID fields **3600***a* and **3640***a* with field **3500***a*). Description field **3600** may provide a useful description last saved by the user for the permission entry. Block **3908** may also retrieve system predefined data records for use and/or management. Thereafter, each joined entry returned at block **3908** is associated at block **3910** with the corresponding data IDs (at least fields **3500***a* and **3540***a*) for easy unique record accesses when the user acts on the data. Block **3910** also initializes a list cursor to point to the first list entry to be presented to the user. Thereafter, block **3912** sets user interface indication for where the list cursor is currently set (e.g. set to highlight the entry), and any list scrolling settings are set (the list is initially not set for being scrolled on first FIG. **39**A processing encounter to block **3912** from block **3910**). Block **3912** continues to block **3914** where the entry list is presented to the user in accordance with the list cursor and list scroll settings managed for presentation at block **3912**. Thereafter, block **3916** waits for user action to the presented list of permissions data and will continue to block **3918** when a user action has been detected. Presentation of the scrollable list preferably presents in an entry format such that an entry contains fields for: DDR **3600** description; GDR owner information, grantor information and grantee information; GRPDR owner information and group name if applicable; and TDR time spec information. Alternate embodiments will present less information, or more information (e.g. GRTDR(s) **3510** and/or PDR(s) **3530** via GADR(s) **3520** joining fields (e.g. **3500***a*, **3510***a*, **3520***b*)).

If block **3918** determines the user selected to set the list cursor to a different entry, then block **3920** sets the list cursor accordingly and processing continues back to block **3912**. Block **3912** always sets for indicating where the list cursor is currently pointed and sets for appropriately scrolling the list if necessary when subsequently presenting the list at block **3914**. If block **3918** determines the user did not select to set the list cursor, then processing continues to block **3922**. If block **3922** determines the user selected to add a permission, then block **3924** accesses a maximum number of permissions

allowed (perhaps multiple maximum values accessed), and block **3926** checks the maximum(s) with the number of current permissions defined. There are many embodiments for what deems a maximum (for this user, for a group, for this MS, etc). If block **3926** determines a maximum number of permissions allowed already exists, then block **3928** provides an error to the user and processing continues back to block **3912**. Block **3928** preferably requires the user to acknowledge the error before continuing back to block **3912**. If block **3926** determines a maximum was not exceeded, then block **3930** interfaces with the user for entering validated permission data and block **3932** adds the data record(s), appropriately updates the list with the new entry, and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **3912**. If block **3922** determines the user did not want to add a permission, processing continues to block **3934**. Block **3932** will add a GDR **3500**, DDR **3600**, HDR **3620** (to set creator information) and TDR **3640**. The DDR and TDR are optionally added by the user, but the DDR may be strongly suggested (if not enforced on the add). This will provide a permission record assigning all privileges from the grantor to the grantee. Additionally, blocks **3930**/**3932** may support adding new GADR(s) **3520** for assigning certain grants and/or privileges (which are validated to exist prior to adding data at block **3932**).

If block **3934** determines the user selected to delete a permission, then block **3936** deletes the data record currently pointed to by the list cursor, modifies the list for the discarded entry, and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **3912**. Block **3936** will use the granting ID field **3500***a* (associated with the entry at block **3910**) to delete the permission. Associated GADR(s) **3520**, DDR **3600**, HDR **3620**, and TDR **3640** is also deleted (e.g. preferably with a cascade delete in a SQL embodiment). If block **3934** determines the user did not select to delete a permission, then processing continues to block **3952** of FIG. **39**B by way of off-page connector **3950**.

With reference now to FIG. **39**B, if block **3952** determines the user selected to modify a permission, then block **3954** interfaces with the user to modify permission data of the entry pointed to by the list cursor. The user may change information of the GDR and any associated records (e.g. DDR, TDR and GADR(s)). The user may also add the associated records at block **3954**. Block **3954** waits for a user action indicating completion. Block **3954** will continue to block **3956** when the complete action is detected at block **3954**. If block **3956** determines the user exited, then processing continues back to block **3912** by way of off-page connector **3998**. If block **3956** determines the user selected to save changes made at block **3954**, then block **3958** updates the data and the list is appropriately updated before continuing back to block **3912**. Block **3958** may update the GDR and/or any associated records (e.g. GADR(s), DDR, and/or TDR) using the permission id field **3500***a* (associated to the entry at block **3910**). Block **3958** will update an associated HDR as well. Block **3958** may add new GADR(s), a DDR and/or TDR as part of the permission change. If block **3952** determines the user did not select to modify a permission, then processing continues to block **3960**.

If block **3960** determines the user selected to get more details of the permission (e.g. show all joinable data to the GDR that is not already presented with the entry), then block **3962** gets additional details (may involve database queries in an SQL embodiment) for the permission pointed to by the list cursor, and block **3964** appropriately presents the information to the user. Block **3964** then waits for a user action that the user is complete reviewing details, in which case processing continues back to block **3912**. If block **3960** determines the user did not select to get more detail, then processing continues to block **3966**.

If block **3966** determines the user selected to internalize permissions data thus far being maintained, then block **3968** internalizes (e.g. as a compiler would) all applicable data records for well performing use by the MS, and block **3970** saves the internalized form, for example to MS high speed non-persistent memory. In one embodiment, blocks **3968** and **3970** internalize permission data to applicable C structures of FIGS. **34**A through **34**G (also see FIG. **52**). In various embodiments, block **3968** maintains statistics for exactly what was internalized, and updates any running totals or averages maintained for a plurality of internalizations up to this point, or over certain time periods. Statistics such as: number of active constructs; number of user construct edits of particular types; amount of associated storage used, freed, changed, etc with perhaps a graphical user interface to graph changes over time; number of privilege types specified, number of charters affected by permissions; and other permission dependent statistics. In other embodiments, statistical data is initialized at internalization time to prepare for subsequent gathering of useful statistics during permission processing. In embodiments where a tense qualifier is specified for TimeSpec information, saving the internalized form at block **3970** causes all past and current tense configurations to become effective for being processed.

Bock **3970** then continues back to block **3912**. If block **3966** determines the user did not select to internalize permission configurations, then processing continues to block **3972**. Alternate embodiments of processing permissions **10** in the present disclosure will rely upon the data records entirely, rather than requiring the user to redundantly internalize from persistent storage to non-persistent storage for use. Persistent storage may be of reasonably fast performance to not require an internalized version of permission **10**. Different embodiments may completely overwrite the internalized form, or update the current internalized form with any changes.

If block **3972** determines the user selected to exit block **3810** processing, then block **3974** cleans up processing thus far accomplished (e.g. issue a stop using database command), and block **3976** completes block **3810** processing. If block **3972** determines the user did not select to exit, then processing continues to block **3978** where all other user actions detected at block **3916** are appropriately handled, and processing continues back to block **3916** by way off off-page connector **3996**.

FIGS. **40**A through **40**B depict flowcharts for describing a preferred embodiment of MS user interface processing for grants configuration of block **3814**. With reference now to FIG. **40**A, processing starts at block **4002**, continues to block **4004** for initialization (e.g. a start using database command), and then to block **4006** where groups the user is a member of are accessed. Block **4006** retrieves all GRPDRs **3540** joined to GADRs **3520** such that the descendant type field **3520***c* and descendant ID field **3520***d* match the user information, and the ascendant type field **3520***a* is set to Group and the ascendant ID field **3520***b* matches the group ID field **3540***a*. While there may be different types of groups as defined for the BNF grammar, the GRPDR **3540** is a derivative embodiment which happens to not distinguish. Alternate embodiments may carry a group type field to select appropriate records by group type. Yet another embodiment may not have a block **4006** with processing at block **4008** for gathering data additionally by groups the user is a member of. Block **4006** continues to block **4008**.

Block **4008** accesses all GRTDRs **3510** (e.g. all rows from a GRTDR SQL table) for the user of FIG. **40**A matching the owner information of the GRTDRs (e.g. user information matches field **3510***b*) to the user and to groups the user is a member of (e.g. group information matches field **3510***b* (e.g. owner type=group, owner id=group ID field **3540***a* from block **4006**). The GRTDRs **3510** are additionally joined (e.g. SQL join) with DDRs **3600** and TDRs **3640** (e.g. fields **3600***b* and **3640***b*=Grant and by matching ID fields **3600***a* and **3640***a* with field **3510***a*). Description field **3600***c* can provide a useful description last saved by the user for the grant data, however the grant name itself is preferably self documenting. Block **4008** may also retrieve system predefined data records for use and/or management. Block **4008** will also retrieve grants within grants to present the entire tree structure for a grant entry. Block **4008** retrieves all GRTDRs **3510** joined to other GRTDRs **3510** through GADRs **3520** which will provide the grant tree structure hierarchy. Grants can be descendant to other grants in a grant hierarchy. Descendant type field **3520***c* set to Grant and descendant ID field **3520***d* for a particular grant will be a descending grant to an ascending grant of ascendant type field **3520***a* set to Grant and ascendant ID field **3520***b*. Therefore, each list entry is a grant entry that may be any node of a grant hierarchy tree. There may be grant information redundantly presented, for example when a grant is subordinate to more than one grant, but this helps the user know a grant tree structure if one has been configured. A visually presented embodiment may take the following form wherein a particular Grant_i appears in the appropriate hierarchy form.

Grant Info_1
- Grant Info_{11}
- Grant Info_{12}
  - Grant Info_{121}
  - Grant Info_{122}
  - ...
  - Grant Info_{12n}
- ...
- Grant Info_{1k}

Grant Info_2
...
Grant Info_j

The list cursor can be pointing to any grant item within a single grant entry hierarchy. Thus, a single grant entry can be represented by a visual nesting, if applicable. Thereafter, each joined entry returned at block **4008** is associated at block **4010** with the corresponding data IDs (at least fields **3510***a* and **3540***a*) for easy unique record accesses when the user acts on the data. Block **4010** also initializes a list cursor to point to the first grant item to be presented to the user in the (possibly nested) list. Thereafter, block **4012** sets user interface indication for where the list cursor is currently set (e.g. set to highlight the entry) and any list scrolling settings are set (the list is initially not set for being scrolled on first FIG. **40**A processing encounter to block **4012** from block **4010**. Block **4012** continues to block **4014** where the entry list is presented to the user in accordance with the list cursor and list scroll settings managed for presentation at block **4012**. Thereafter, block **4016** waits for user action to the presented list of grant data and will continue to block **4018** when a user action has been detected. Presentation of the scrollable list preferably presents in an entry format with subordinate grants also reference-able by the list cursor. A grant entry of the grant tree presented preferably contains fields for: GRTDR name field **3510***c*; GRTDR owner information; GRPDR owner information and group name if applicable; TDR time spec information; and DDR information. Alternate embodiments will present less information, or more information (e.g. join PDR(s) **3530** via GADR(s) **3520** when applicable).

If block **4018** determines the user selected to set the list cursor to a different grant reference, then block **4020** sets the list cursor accordingly and processing continues back to block **4012**. Block **4012** always sets for indicating where the list cursor is currently pointed and sets for appropriately scrolling the list if necessary when subsequently presenting the list at block **4014**. If block **4018** determines the user did not select to set the list cursor, then processing continues to block **4022**. If block **4022** determines the user selected to add a grant, then block **4024** accesses a maximum number of grants allowed (perhaps multiple maximum values accessed), and block **4026** checks the maximum(s) with the number of current grants defined. There are many embodiments for what deems a maximum (for this user, for a group, for this MS, etc). If block **4026** determines a maximum number of grants allowed already exists, then block **4028** provides an error to the user and processing continues back to block **4012**. Block **4028** preferably requires the user to acknowledge the error before continuing back to block **4012**. If block **4026** determines a maximum was not exceeded, then block **4030** interfaces with the user for entering validated grant data and block **4032** adds the data record, appropriately updates the list with the new entry, and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **4012**. If block **4022** determines the user did not want to add a grant, processing continues to block **4034**. Block **4032** will add a GRTDR **3500**, DDR **3600**, HDR **3620** (to set creator information) and TDR **3640**. The DDR and TDR are optionally added by the user. Additionally, at block **4030** the user may add new GADR(s) **3520** for assigning certain grants to the added grant and/or privileges to the grant (which are validated to exist prior to adding data at block **4032**).

If block **4034** determines the user selected to modify a grant, then block **4036** interfaces with the user to modify grant data of the entry pointed to by the list cursor. The user may change information of the GRTDR and any associated records (e.g. DDR, TDR and GADR(s)). The user may also add the associated records at block **4036**. Block **4036** waits for a user action indicating completion. Block **4036** will continue to block **4038** when the action is detected at block **4036**. If block **4038** determines the user exited, then processing continues back to block **4012**. If block **4038** determines the user selected to save changes made at block **4036**, then block **4040** updates the data and the list is appropriately updated before continuing back to block **4012**. Block **4040** may update the GRTDR and/or any associated records (e.g. GADR(s), DDR, and/or TDR) using the grant id field **3510***a* (associated to the grant item at block **4010**). Block **4040** will update an associated HDR as well. Block **4036** may add new GADR(s), a DDR and/or TDR as part of the grant change. If block **4034** determines the user did not select to modify a grant, then processing continues to block **4052** by way of off-page connector **4050**.

With reference now to FIG. **40**B, if block **4052** determines the user selected to get more details of the grant (e.g. show all joinable data to the GRTDR that is not already presented with the entry), then block **4054** gets additional details (may involve database queries in an SQL embodiment) for the grant pointed to by the list cursor, and block **4056** appropriately presents the information to the user. Block **4056** then waits for a user action that the user is complete reviewing details, in which case processing continues back to block **4012** by way

of off-page connector **4098**. If block **4052** determines the user did not is select to get more detail, then processing continues to block **4058**.

If block **4058** determines the user selected to delete a grant, then block **4060** determines any data records (e.g. GADR(s) **3520**) that reference the grant data record to be deleted. Preferably, no ascending data records (e.g. GRTDRs) are joinable to the grant data record being deleted, otherwise the user may improperly delete a grant from a configured permission or other grant. In the case of descending grants, all may be cascaded deleted in one embodiment, provided no ascending grants exist for any of the grants to be deleted. The user should remove ascending references to a grant for deletion first. Block **4060** continues to block **4062**. If block **4062** determines there was at least one reference, block **4064** provides an appropriate error with the reference(s) found so the user can subsequently reconcile. Block **4064** preferably requires the user to acknowledge the error before continuing back to block **4012**. If no references were found as determined by block **4062**, then processing continues to block **4066** for deleting the data record currently pointed to by the list cursor, along with any other related records that can be deleted. Block **4066** also modifies the list for the discarded entry(s), and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **4012**. Block **4066** will use the grant ID field **3510***a* (associated with the entry at block **4010**) to delete a grant. Associated records (e.g. DDR **3600**, HDR **3620**, and TDR **3640**) are also deleted (e.g. preferably with a cascade delete in a SQL embodiment). If block **4058** determines the user did not select to delete a grant, then processing continues to block **4068**.

If block **4068** determines the user selected to exit block **3814** processing, then block **4070** cleans up processing thus far accomplished (e.g. issue a stop using database command), and block **4072** completes block **3814** processing. If block **4068** determines the user did not select to exit, then processing continues to block **4074** where all other user actions detected at block **4016** are appropriately handled, and processing continues back to block **4016** by way off off-page connector **4096**.

FIGS. **41**A through **41**B depict flowcharts for describing a preferred embodiment of MS user interface processing for groups configuration of block **3818**. With reference now to FIG. **41**A, processing starts at block **4102**, continues to block **4104** for initialization (e.g. a start using database command), and then to block **4106** where groups the user is a member of are accessed. Block **4106** retrieves all GRPDRs **3540** joined to GADRs **3520** such that the descendant type field **3520***c* and descendant ID field **3520***d* match the user information, and the ascendant type field **3520***a* is set to Group and the ascendant ID field **3520***b* matches the group ID field **3540***a*. While there may be different types of groups as defined for the BNF grammar, the GRPDR **3540** is a derivative embodiment which happens to not distinguish. Alternate embodiments may carry a group type field to select appropriate records by group type. Yet another embodiment may not have a block **4106** with processing at block **4108** for gathering data additionally by groups the user is a member of. Block **4106** continues to block **4108**.

Block **4108** accesses all GRPDRs **3540** (e.g. all rows from a GRPDR SQL table) for the user of FIG. **41**A matching the owner information of the GRPDRs (e.g. user information matches field **3540***b*) to the user and to groups the user is a member of (e.g. group information matches field **3540***b* (e.g. owner type=group, owner id=group ID field **3540***a* from block **4106**). The GRPDRs **3540** are additionally joined (e.g. SQL join) with DDRs **3600** and TDRs **3640** (e.g. fields **3600***b* and **3640***b*=Group and by matching ID fields **3600***a* and **3640***a* with field **3540***a*). Description field **3600***c* can provide a useful description last saved by the user for the group data, however the group name itself is preferably self documenting. Block **4108** may also retrieve system predefined data records for use and/or management. Block **4108** will also retrieve groups within groups to present the entire tree structure for a group entry. Block **4108** retrieves all GRPDRs **3540** joined to other GRPDRs **3540** through GADRs **3520** which will provide the group tree structure hierarchy. Groups can be descendant to other groups in a group hierarchy. Descendant type field **3520***c* set to Group and descendant ID field **3520***d* for a particular group will be a descending group to an ascending group of ascendant type field **3520***a* set to Group and ascendant ID field **3520***b*. Therefore, each list entry is a group entry that may be any node of a group hierarchy tree. There may be group information redundantly presented, for example when a group is subordinate to more than one group, but this helps the user know a group tree structure if one has been configured. A visually presented embodiment may take the following form wherein a particular Group_i appears in the appropriate hierarchy form.

Group Info_1
- Group Info_{11}
- Group Info_{12}
  - Group Info_{121}
  - Group Info_{122}
  - ...
  - Group Info_{12u}
- ...
- Group Info_{1t}

Group Info_2
...
Group Info_s

The list cursor can be pointing to any group item within a single group entry hierarchy. Thus, a single group entry can be represented by a visual nesting, if applicable. Thereafter, each joined entry returned at block **4108** is associated at block **4110** with the corresponding data IDs (at least fields **3540***a*) for easy unique record accesses when the user acts on the data. Block **4110** also initializes a list cursor to point to the first group item to be presented to the user in the (possibly nested) list. Thereafter, block **4112** sets user interface indication for where the list cursor is currently set (e.g. set to highlight the entry) and any list scrolling settings are set (the list is initially not set for being scrolled on first FIG. **41**A processing encounter to block **4112** from block **4110**). Block **4112** continues to block **4114** where the entry list is presented to the user in accordance with the list cursor and list scroll settings managed for presentation at block **4112**. Thereafter, block **4116** waits for user action to the presented list of group data and will continue to block **4118** when a user action has been detected. Presentation of the scrollable list preferably presents in an entry format with subordinate groups also reference-able by the list cursor. A group entry of the group tree presented preferably contains fields for: GRPDR name field **3540***c*; GRPDR owner information; owning GRPDR owner information and group name if applicable; TDR time spec information; and DDR information. Alternate embodiments will present less information, or more information (e.g. join to specific identities via GADR(s) **3520** when applicable).

If block **4118** determines the user selected to set the list cursor to a different group entry, then block **4120** sets the list cursor accordingly and processing continues back to block **4112**. Block **4112** always sets for indicating where the list

cursor is currently pointed and sets for appropriately scrolling the list if necessary when subsequently presenting the list at block **4114**. If block **4118** determines the user did not select to set the list cursor, then processing continues to block **4122**. If block **4122** determines the user selected to add a group, then block **4124** accesses a maximum number of groups allowed (perhaps multiple maximum values accessed), and block **4126** checks the maximum(s) with the number of current groups defined. There are many embodiments for what deems a maximum (for this user, for a group, for this MS, etc). If block **4126** determines a maximum number of groups allowed already exists, then block **4128** provides an error to the user and processing continues back to block **4112**. Block **4128** preferably requires the user to acknowledge the error before continuing back to block **4112**. If block **4126** determines a maximum was not exceeded, then block **4130** interfaces with the user for entering validated group data and block **4132** adds the data record, appropriately updates the list with the new entry, and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **4112**. If block **4122** determines the user did not want to add a group, processing continues to block **4134**. Block **4132** will add a GRTDR **3500**, DDR **3600**, HDR **3620** (to set creator information) and TDR **3640**. The DDR and TDR are optionally added by the user. Additionally, at block **4130** the user may add new GADR(S) **3520** for assigning certain groups to the added group and/or identities to the group (which are validated to exist prior to adding data at block **4132**).

If block **4134** determines the user selected to modify a group, then block **4136** interfaces with the user to modify group data of the entry pointed to by the list cursor. The user may change information of the GRPDR and any associated records (e.g. DDR, TDR and GADR(s)). The user may also add the associated records at block **4136**. Block **4136** waits for a user action indicating completion. Block **4136** will continue to block **4138** when the complete action is detected at block **4136**. If block **4138** determines the user exited, then processing continues back to block **4112**. If block **4138** determines the user selected to save changes made at block **4136**, then block **4140** updates the data and the list is appropriately updated before continuing back to block **4112**. Block **4140** may update the GRPDR and/or any associated GADR(s), DDR, and/or TDR using the group id field **3540***a* associated to the group item at block **4110**. Block **4140** will update an associated HDR as well. Blocks **4136**/**4140** may support adding new GADR(s), a DDR and/or TDR as part of the group change. If block **4134** determines the user did not select to modify a group, then processing continues to block **4152** by way of off-page connector **4150**.

With reference now to FIG. **41**B, if block **4152** determines the user selected to get more details of the group (e.g. show all joinable data to the GRPDR that is not already presented with the entry), then block **4154** gets additional details (may involve database queries in an SQL embodiment) for the group pointed to by the list cursor, and block **4156** appropriately presents the information to the user. Block **4156** then waits for a user action that the user is complete reviewing details, in which case processing continues back to block **4112** by way of off-page connector **4198**. If block **4152** determines the user did not select to get more detail, then processing continues to block **4158**.

If block **4158** determines the user selected to delete a group, then block **4160** determines any data records (e.g. GADR(s) **3520**) that reference the group data record to be deleted. Preferably, no ascending data records (e.g. GRPDRs) are joinable to the group data record being deleted, otherwise the user may improperly delete a group from a configured permission or other group. In the case of descending groups, all may be cascaded deleted in one embodiment, provided no ascending groups exist for any of the groups to be deleted. The user should remove ascending references to a group for deletion first. Block **4160** continues to block **4162**. If block **4162** determines there was at least one reference, block **4164** provides an appropriate error with the reference(s) found so the user can subsequently reconcile. Block **4164** preferably requires the user to acknowledge the error before continuing back to block **4112**. If no references were found as determined by block **4162**, then processing continues to block **4166** for deleting the data record currently pointed to by the list cursor, along with any other related records that can be deleted. Block **4166** also modifies the list for the discarded entry(s), and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **4112**. Block **4166** will use the group ID field **3540***a* (associated with the entry at block **4110**) to delete the group. Associated records (e.g. DDR **3600**, HDR **3620**, and TDR **3640**) are also deleted (e.g. preferably with a cascade delete in a SQL embodiment). If block **4158** determines the user did not select to delete a group, then processing continues to block **4168**.

If block **4168** determines the user selected to exit block **3818** processing, then block **4170** cleans up processing thus far accomplished (e.g. issue a stop using database command), and block **4172** completes block **3818** processing. If block **4168** determines the user did not select to exit, then processing continues to block **4174** where all other user actions detected at block **4116** are appropriately handled, and processing continues back to block **4116** by way off off-page connector **4196**.

FIG. **42** depicts a flowchart for describing a preferred embodiment of a procedure for viewing MS configuration information of others. Processing starts at block **4202** and continues to block **4204** where an object type parameter is determined for which information to present to the user as passed by the caller of FIG. **42** processing (e.g. GROUP_INFO, PERMISSION_INFO, GRANT_INFO, CHARTER_INFO, ACTION_INFO or PARAMETER_INFO). Thereafter, block **4206** performs initialization (e.g. a start using database command), and then the user specifies owner information (criteria), at block **4208**, for the object type data records to present. No privilege is assumed required for browsing other's information since it is preferably local to the MS of the user anyway. Block **4208** continues to block **4210**.

In an alternative embodiment, block **4208** appropriately accesses privileges granted from the owner criteria to the user of FIG. **42** to ensure the user has a privilege to browse the data records (per object type parameter) of the specified owner. Block **4208** will provide an error when there is no privilege, and will continue to block **4210** when there is a privilege. Block **4208** may also provide a user exit option for continuing to block **4216** for cases the user cannot successfully specify owner criteria. In similar embodiments, there may be a separate privilege required for each object type a user may browse.

Block **4210** gets (e.g. SQL selects) data according to the object type parameter (e.g. GRPDR(s), GDR(s), GRTDR(s), CDR(s), ADR(s) or PARMDR(s), along with any available associated joinable data (e.g. DDR(s), HDR(s), TDR(s) and data records via GADR(s) if applicable), per object type passed). There are various embodiments to block **4210** in accessing data: locally maintained data for the owner criteria specified at block **4208**, communicating with a remote MS for accessing the MS of the owner criteria to synchronously pull the data, or sending a request to a remote MS over an interface like interface **1926** for then asynchronously receiving by an

interface like interface **1948** for processing. One preferred embodiment is to locally maintain relevant data. In privilege enforced embodiments, appropriate privileges are determined before allowing access to the other's data.

Thereafter, if block **4212** determines there were no data records according to the object type passed by the caller for the owner criteria specified at block **4208**, then block **4214** provides an error to the user, and processing continues to block **4216**. Block **4216** performs cleanup of processing thus far accomplished (e.g. perform a stop using database command), and then continues to block **4218** for returning to the caller of FIG. **42** processing. Block **4214** preferably requires the user to acknowledge the error before continuing to block **4216**.

If block **4212** determines at least one data record of object type was found, then block **4220** presents a browse-able scrollable list of entries to the user (i.e. similar to lists discussed for presentation by FIGS. **39**A&B, FIGS. **40**A&B, FIGS. **41**A&B, FIGS. **46**A&B, FIGS. **47**A&B or FIGS. **48**A&B, per object typed passed), and block **4222** waits for a user action in response to presenting the list. When a user action is detected at block **4222**, processing continues to block **4224**. If block **4224** determines the user selected to specify new owner criteria (e.g. for comparison to field **3500***b*, **3510***b*, **3540***b*, **3700***b*, **3750***b* or **3775***b*, per object type passed) for browse, then processing continues back to block **4208** for new specification and applicable processing already discussed for blocks thereafter. If block **4224** determines the user did not select to specify new owner criteria, processing continues to block **4226**.

If block **4226** determines the user selected to get more detail of a selected list entry, then processing continues to block **4228** for getting data details of the selected entry, and block **4230** presents the details to the user, and waits for user action. Detail presentation is similar to getting detail processing discussed for presentation by FIGS. **39**A&B, FIGS. **40**A&B, FIGS. **41**A&B, FIGS. **46**A&B, FIGS. **47**A&B or FIGS. **48**A&B, per object typed passed. Block **4230** continues to block **4232** upon a user action (complete/clone).

If block **4232** determines the user action from block **4230** was to exit browse, processing continues to block **4220**. If block **4232** determines the user action from block **4230** was to clone the data (e.g. to make a copy for user's own use), processing continues to block **4234** for accessing permissions. Thereafter, if block **4236** determines the user does not have permission to clone, processing continues to block **4238** for reporting an error (preferably requiring the user to acknowledge before leaving block **4238** processing), and then back to block **4220**. If block **4236** determines the user does have permission to clone, processing continues to block **4240** where the data item browsed is appropriately duplicated with defaulted fields as though the user of FIG. **42** processing had created new data himself. Processing then continues back to block **4220**. If block **4226** determines the user did not select to get more detail on a selected item, then processing continues to block **4242**.

If block **4242** determines the user selected to exit browse processing, then processing continues to block **4216** already described. If block **4242** determines the user did not select to exit, then processing continues to block **4244** where all other user actions detected at block **4222** are appropriately handled, and processing continues back to block **4222**.

In an alternate embodiment, FIG. **42** will support cloning multiple entries in one action so that a first user conveniently makes use of a second user's data (like starter template(s)) for the first user to create/configure new data without entering it from scratch in the other interfaces disclosed. Another embodiment will enforce unique privileges for which data can be cloned by which user(s).

FIG. **43** depicts a flowchart for describing a preferred embodiment of a procedure for configuring MS acceptance of data from other MSs, for example permissions **10** and charters **12**. In a preferred embodiment, permissions **10** and charters **12** contain data for not only the MS **2** but also other MSs which are relevant to the MS **2** (e.g. MS users are known to each other). Processing starts at block **4302** and continues to block **4304** where a parameter passed by a caller is determined. The parameter indicates which object type (data type) to configure delivery acceptance (e.g. PERMISSION_INFO, CHARTER_INFO). Thereafter, block **4306** displays acceptable methods for accepting data from other MSs, preferably in a radio button form in a visually perceptible user interface embodiment. A user is presented with two (2) main sets of options, the first set preferably being an exclusive selection:

- Accept no data (MS will not accept data from any source); or
- Accept all data (MS will accept data from any source); or
- Accept data according to permissions (MS will accept data according to those sources which have permission to send certain data (perhaps privilege also specifies by a certain method) to the MS).

And the second set being:

- Targeted data packet sent or broadcast data packet sent (preferably one or the other);
- Electronic Mail Application;
- SMS message; and/or
- Persistent Storage Update (e.g. file system).

Block **4306** continues to block **4308** where the user makes a selection in the first set, and any number of selections in the second set. Thereafter, processing at block **4310** saves the user's selections for the object type parameter passed, and processing returns to the caller at block **4312**. LBX processing may have intelligence for an hierarchy of attempts such as first trying to send or broadcast, if that fails send by email, if that fails send by SMS message, and if that fails alert the MS user for manually copying over the data at a future time (e.g. when MSs are in wireless vicinity of each other). Block **4306** may provide a user selectable order of the attempt types. Intelligence can be incorporated for knowing which data was sent, when it was sent, and whether or not all of the send succeeded, and a synchronous or asynchronous acknowledgement can be implemented to ensure it arrived safely to destination(s). Applicable information is preferably maintained to LBX history **30** for proper implementation.

In one embodiment, the second set of configurations is further governed by individual privileges (each send type), and/or privileges per a source identity. For example, while configurations of the second set may be enabled, the MS will only accept data in a form from a source in accordance with a privilege which is enabled (set for the source identity). Privilege examples (may also each have associated time specification) include:

- Grant Joe privilege to send all types of data (e.g. charters and privileges, or certain (e.g. types, contents, features, any characteristic(s)) charters and/or privileges);
- Grant Joe privilege to send certain type of data (e.g. charters or privileges, or certain (e.g. types, contents, features, any characteristic(s)) charters and/or privileges);
- Grant Joe privilege to send certain type of data using certain method (privilege for each data type and method combination); and/or
- Grant Joe privilege to send certain type of data using certain method(s) (privilege for each data type and method combination) at certain time(s).

In another embodiment, there may be other registered applications (e.g. specified other email applications) which are candidates in the second set. This allows more choices for a receiving application with an implied receiving method (or user may specify an explicit method given reasonable choices of the particular application). For example, multiple MS instant messaging and/or email applications may be selectable in the second set of choices, and appropriately interfaced to for accepting data from other MSs. This allows specifying preferred delivery methods for data (e.g. charters and/or permissions data), and an attempt order thereof.

In some embodiments, charter data that is received may be received by a MS in a deactivated form whereby the user of the receiving MS must activate the charters for use (e.g. define a new charter enabled field **3700***e* for indicating whether or not the charter is active (Y=Yes, N=No)). New field **3700***e* may also be used by the charter originator for disabling or enabling for a variety of reasons. This permits a user to examine charters, and perhaps put them to a test, prior to putting them into use. Other embodiments support activating charters (received and/or originated): one at a time, as selected sets by user specified criteria (any charter characteristic(s)), all or none, by certain originating user(s), by certain originating MS(s), or any other desirable criteria. Of course, privileges are defined for enabling accepting privileges or charters from a MS, but many privileges can be defined for accepting privileges or charters with certain desired characteristics from a MS.

FIG. **44**A depicts a flowchart for describing a preferred embodiment of a procedure for sending MS data to another MS. FIG. **44**A processing is preferably of linkable PIP code **6**. The purpose is for the MS of FIG. **44**A processing (e.g. a first, or sending, MS) to transmit information to other MSs (e.g. at least a second, or receiving, MS), for example permissions **10** or charters **12**. Multiple channels for sending, or broadcasting should be isolated to modular send processing (feeding from a queue **24**). In an alternative embodiment having multiple transmission channels visible to processing of FIG. **44**A (e.g. block **4430**), there can be intelligence to drive each channel for broadcasting on multiple channels, either by multiple send threads for FIG. **44**A processing, FIG. **44**A loop processing on a channel list, and/or passing channel information to send processing feeding from queue **24**. If FIG. **44**A does not transmit directly over the channel(s) (i.e. relies on send processing feeding from queue **24**), an embodiment may provide means for communicating the channel for broadcast/send processing when interfacing to queue **24** (e.g. incorporate a channel qualifier field with send packet inserted to queue **24**).

In any case, see detailed explanations of FIGS. **13**A through **13**C, as well as supporting exemplifications shown in FIGS. **50**A through **50**C, respectively. Processing begins at block **4402**, continues to block **4404** where the caller parameter passed to FIG. **44**A processing is determined (i.e. OBJ_TYPE), and processing continues to block **4406** for interfacing with the user to specify targets to send data to, in context of the object type parameter specified for sending (PERMISSION_INFO or CHARTER_INFO). An alternate embodiment will consult a configuration of data for validated target information. Depending on the present disclosure embodiment, a user may specify any reasonable supported (ID/IDType) combination of the BNF grammar ID construct (see FIG. **30**B) as valid targets. Validation will validate at least syntax of the specification. In another embodiment, block **4406** will access and enforce known permissions for validating which target(s) (e.g. grantor(s)) can be specified. Various embodiments will also support wildcarding the specifications for a group of ID targets (e.g. department* for all department groups). Additional target information is to be specified when required for sending, for example, if email or SMS message is to be used as a send method (i.e. applicable destination recipient addresses to be specified). An alternate embodiment to block **4406** accesses mapped delivery addresses from a database, or table, (referred to as a Recipient Address Book (RAB)) associating a recipient address to a target identity, thereby alleviating the user from manual specification, and perhaps allowing the user to save to the RAB for any new useful RAB data. In another embodiment, block **4428** (discussed below) accesses the RAB for a recipient address for the target when preparing the data for sending.

Upon validation at block **4406**, processing continues to block **4408**. It is possible the user was unsuccessful in specifying targets, or wanted to exit block **4406** processing. If block **4408** determines the user did not specify at least one validated target (equivalent to selecting to exit FIG. **44**A processing), then processing continues to block **4444** where processing returns to the caller. If block **4408** determines there is at least one target specified, then block **4410** accesses LBX history **30** to determine if any of the targets have been sent the specific data already. Thereafter, if block **4412** determines the most recently updated data for a target has already been sent, then block **4414** presents an informative error to the user, preferably requiring user action. Block **4414** continues to block **4416** when the user performs the action. If block **4416** determines the user selected to ignore the error, then processing continues to block **4418**, otherwise processing continues back to block **4406** for updating target specifications.

Block **4418** interfaces with the user to specify a delivery method. Preferably, there are defaulted setting(s) based on the last time the user encountered block **4418**. Any of the "second set" of options described with FIG. **43** can be made. Thereafter, block **4420** logs to LBX history **30** the forthcoming send attempt and gets the next target from block **4406** specifications before continuing to block **4422**. If block **4422** determines that all targets have not been processed, then block **4424** determines applicable OBJ_TYPE data for the target (e.g. check LBX history **30** for any new data that was not previously successfully sent), and block **4426** gets (e.g. preferably new data, or all, depending on embodiment) the applicable target's OBJ_TYPE data (permissions or charters) before continuing to block **4428**. Block **4428** formats the data for sending in accordance with the specified delivery method, along with necessary packet information (e.g. source identity, wrapper data, etc) of this loop iteration (from block **4418**), and block **4430** sends the data appropriately. For a broadcast send, block **4430** broadcasts the information (using a send interface like interface **1906**) by inserting to queue **24** so that send processing broadcasts data **1302** (e.g. on all available communications interface(s) **70**), for example as far as radius **1306**, and processing continues to block **4432**. The broadcast is for reception by data processing systems (e.g. MSs) in the vicinity (see FIGS. **13**A through **13**C, as further explained in detail by FIGS. **50**A through **50**C which includes potentially any distance). For a targeted send, block **4430** formats the data intended for recognition by the receiving target. Block **4430** causes sending/broadcasting data **1302** containing CK **1304**, depending on the type of MS, wherein CK **1304** contains information appropriately. In a send email embodiment, confirmation of delivery status may be used to confirm delivery with an email interface API to check the COD (Confirmation of Delivery) status, or the sending of the email (also SMS message) is assumed to have been delivered in one preferred embodiment.

In an embodiment wherein usual MS communications data **1302** of the MS is altered to contain CK **1304** for listening MSs in the vicinity, send processing feeding from queue **24**, caused by block **4430** processing, will place information as CK **1304** embedded in usual data **1302** at the next opportune time of sending usual data **1302**. This embodiment will replace synchronous sending success validation of blocks **4432** through **4440** and multiple delivery methods of **4418** (and subsequent loop processing) with status asynchronously updated by the receiving MS(s) for a single type of delivery method selected at block **4418**. An alternate embodiment will attempt the multiple send types in an appropriate asynchronous thread of processing depending on success of a previous attempt. As the MS conducts its normal communications, transmitted data **1302** contains new data CK **1304** to be ignored by receiving MS other character **32** processing, but to be found by listening MSs within the vicinity which anticipate presence of CK **1304**. Otherwise, when LN-Expanse deployments have not introduced CK **1304** to usual data **1302** communicated on a receivable signal by MSs in the vicinity, FIG. **44**A sends/broadcasts new data **1302**.

For sending an email, SMS message, or other application delivery method, block **4430** will use the additional target information (recipient address) specified via block **4406** for properly sending. Thereafter, block **4432** waits for a synchronous acknowledgement if applicable before either receiving one or timing out. If a broadcast was made, one (1) acknowledgement may be all that is necessary for validation, or all anticipated targets can be accounted for before deeming a successful ack. An email, SMS message, or other application send may be assumed reliable and that an ack was received. Thereafter, if block **4434** determines an applicable ack was received (i.e. data successfully sent/received), or none was anticipated (i.e. assume got it), then processing continues back to block **4420** for processing any next target(s). If block **4434** determines an anticipated ack was not received, then block **4436** logs the situation to LBX history **30** and the next specified delivery method is accessed. Thereafter, if block **4438** determines all delivery methods have already been processed for the current target, then processing continues to block **4440** for logging the overall status and providing an error to the user. Block **4440** may require a user acknowledgement before continuing back to block **4420**. If block **4438** determines there is another specified delivery method for sending, then processing continues back to block **4428** for sending using the next method.

Referring back to block **4422**, if all targets are determined to have been processed, then block **4442** maintains FIG. **44**A processing results to LBX history **30** and the caller is returned to at block **4444**. In an alternate embodiment to FIG. **44**A processing, a trigger implementation is used for sending/broadcasting data at the best possible time (e.g. when new/modified permissions or charters information is made for a target) as soon as possible, as soon as a target is detected to be nearby, or in the vicinity (vicinity is expanded as explained by FIGS. **50**A through **50**C), or as soon as the user is notified to send (e.g. in response to a modification) and then acknowledges to send. See FIGS. **50**A through **50**C for explanation of communicating data from a first MS to a second MS over greater distances. In another embodiment, background thread(s) timely poll (e.g. per user or system configurations) the permissions and/or charters data to determine which data should be sent, how to send it, who to send it to, what applicable permissions are appropriate, and when the best time is to send it. A time interval, or schedule, for sending data to others on a continual interim basis may also be configured. This may be particularly useful as a user starts using a MS for the first time and anticipates making many configuration changes. The user may start or terminate polling threads as part of FIGS. **14**A/**14**B processing, so that FIG. **44**A is relied on to make sure permissions and/or charters are communicated as needed. Appropriate blocks of FIGS. **44**A&B will also interface to statistics **14** for reporting successes, failures and status of FIGS. **44**A&B processing.

In sum, FIGS. **44**A and **44**B provide a LBX peer to peer method for ensuring permissions and charters are appropriately maintained at MSs, wherein FIG. **44**A sends in a peer to peer fashion and FIG. **44**B receives in a peer to peer to fashion. Thus, permissions **10** and charters **12** are sent from a first MS to a second MS for configuring maintaining, enforcing, and/or processing permissions **10** and charters **12** at an MS. There is no intermediary service required for permissions and charters for LBX interoperability. FIG. **44**A demonstrates a preferred push model. A pull model may be alternatively implemented. An alternative embodiment may make a request to a MS for its permissions and/or charters and then populate its local image of the data after receiving the response. Privileges would be appropriately validated at the sending MS(s) and/or receiving MS(s) in order to ensure appropriate data is sent/received to/from the requesting MS.

FIG. **44**B depicts a flowchart for describing a preferred embodiment of receiving MS configuration data from another MS. FIG. **44**B processing describes a Receive Configuration Data (RxCD) process worker thread, and is of PIP code **6**. There may be many worker threads for the RxCD process, just as described for a **19***xx* process. The receive configuration data (RxCD) process is to fit identically into the framework of architecture **1900** as other **19***xx* processes, with specific similarity to process **1942** in that there is data received from receive queue **26**, the RxCD thread(s) stay blocked on the receive queue until data is received, and a RxCD worker thread sends data as described (e.g. using send queue **24**). Blocks **1220** through **1240**, blocks **1436** through **1456** (and applicable invocation of FIG. **18**), block **1516**, block **1536**, blocks **2804** through **2818**, FIG. **29**A, FIG. **29**B, and any other applicable architecture **1900** process/thread framework processing is to adapt for the new RxCD process. For example, the RxCD process is initialized as part of the enumerated set at blocks **1226** (preferably last member of set) and **2806** (preferably first member of set) for similar architecture **1900** processing. Receive processing identifies targeted/broadcasted data (permissions and/or charter data) destined for the MS of FIG. **44**B processing. An appropriate data format is used, for example the X.409 encoding of FIGS. **33**A through **33**C wherein RxCD thread(s) purpose is for the MS of FIG. **44**B processing to respond to incoming data. It is recommended that validity criteria set at block **1444** for RxCD-Max be set as high as possible (e.g. 10) relative performance considerations of architecture **1900**, to service multiple data receptions simultaneously. Multiple channels for receiving data fed to queue **26** are preferably isolated to modular receive processing.

In an alternative embodiment having multiple receiving transmission channels visible to the RxCD process, there can be a RxCD worker thread per channel to handle receiving on multiple channels simultaneously. If RxCD thread(s) do not receive directly from the channel, the preferred embodiment of FIG. **44**B would not need to convey channel information to RxCD thread(s) waiting on queue **24** anyway. Embodiments could allow specification/configuration of many RxCD thread(s) per channel.

A RxCD thread processing begins at block **4452** upon the MS receiving permission data and/or charter data, continues to block **4454** where the process worker thread count RxCD-

Ct is accessed and incremented by 1 (using appropriate semaphore access (e.g. RxCD-Sem)), and continues to block **4456** for retrieving from queue **26** sent data (using interface like interface **1948**), perhaps a special termination request entry, and only continues to block **4458** when a record of data (permission/charter data, or termination record) is retrieved. In one embodiment, receive processing deposits X.409 encoding data as record(s) to queue **26**, and may break up a datastream into individual records of data from an overall received (or ongoing) datastream. In another embodiment, XML is received and deposited to queue **26**, or some other suitable syntax is received as derived from the BNF grammar. In another embodiment, receive processing receives data in one format and deposits a more suitable format for FIG. **44**B processing. Receive processing embodiments may deposit "piece-meal" records of data as sent, "piece-meal" records broken up from data received, full charter or permission datastreams and/or subsets thereof to queue **26** for processing by FIG. **44**B.

Block **4456** stays blocked on retrieving from queue **26** until any record is retrieved, in which case processing continues to block **4458**. If block **4458** determines a special entry indicating to terminate was not found in queue **26**, processing continues to block **4460**. There are various embodiments for RxCD thread(s), thread(s) **1912** and thread(s) **1942** to feed off a queue **26** for different record types, for example, separate queues **26**A, **26**B and **26**C, or a thread target field with different record types found at queue **26** (e.g. like field **2400***a*). In another embodiment, there are separate queues **26**C and **26**D for separate processing of incoming charter and permission data. In another embodiment, thread(s) **1912** are modified with logic of RxCD thread(s) to handle permission and/or charter data records, since thread(s) **1912** are listening for queue **26** data anyway. In another embodiment, there are segregated RxCD threads RxCD-P and RxCD-C for separate permission and charter data processing.

Block **4460** validates incoming data for this targeted MS before continuing to block **4462**. A preferred embodiment of receive processing already validated the data is intended for this MS by having listened specifically for the data, or by having already validated it is at the intended MS destination (e.g. block **4458** can continue directly to block **4464** (no block **4460** and block **4462** required)). If block **4462** determines the data is valid for processing, then block **4464** accesses the data source identity information (e.g. owner information, sending MS information, grantor/grantee information, etc, as appropriate for an embodiment), block **4466** accesses acceptable delivery methods and/or permissions/privileges for the source identity to check if the data is eligible for being received, and block **4468** checks the result. Depending on an embodiment, block **4466** may enforce an all or none privilege for accepting the privilege or charter data, or may enforce specific privileges from the receiving MS (MS user) to the sending MS (MS user) for exactly which privileges or charters are acceptable to be received and locally maintained.

If block **4468** determines the delivery is acceptable (and perhaps privileged, or privileged per source), then block **4470** appropriately updates the MS locally with the data (depending on embodiment of **4466**, block **4470** may remove from existing data at the MS as well as per privilege(s)), block **4472** completes an acknowledgment, and block **4474** sends/broadcasts the acknowledgement (ack), before continuing back to block **4456** for more data. Block **4474** sends/broadcasts the ack (using a send interface like interface **1946**) by inserting to queue **24** so that send processing transmits data **1302**, for example as far as radius **1306**. Embodiments will use the different correlation methods already discussed above, to associate an ack with a send.

If block **4468** determines the data is not acceptable, then processing continues directly back to block **4456**. For security reasons, it is best not to respond with an error. It is best to ignore the data entirely. In another embodiment, an error may be returned to the sender for appropriate error processing and reporting. Referring back to block **4462**, if it is determined that the data is not valid, then processing continues back to block **4456**.

Referring back to block **4458**, if a worker thread termination request was found at queue **26**, then block **4476** decrements the RxCD worker thread count by 1 (using appropriate semaphore access (e.g. RxCD-Sem)), and RxCD thread processing terminates at block **4478**. Block **4476** may also check the RxCD-Ct value, and signal the RxCD process parent thread that all worker threads are terminated when RxCD-Ct equals zero (0).

Block **4474** causes sending/broadcasting data **1302** containing CK **1304**, depending on the type of MS, wherein CK **1304** contains ack information prepared. In the embodiment wherein usual MS communications data **1302** of the MS is altered to contain CK **1304** for listening MSs in the vicinity, send processing feeding from queue **24**, caused by block **4474** processing, will place ack information as CK **1304** embedded in usual data **1302** at the next opportune time of sending usual data **1302**. As the MS conducts its normal communications, transmitted data **1302** contains new data CK **1304** to be ignored by receiving MS other character **32** processing, but to be found by listening MSs within the vicinity which anticipate presence of CK **1304**. Otherwise, when LN-Expanse deployments have not introduced CK **1304** to usual data **1302** communicated on a receivable signal by MSs in the vicinity, FIG. **44**B sends/broadcasts new ack data **1302**.

In an alternate embodiment, permission and/or charter data records contain a sent date/time stamp field of when the data was sent by a remote MS, and a received date/time stamp field (like field **2490***c*) is processed at the MS in FIG. **44**B processing. This would enable calculating a TDOA measurement while receiving data (e.g. permissions and/or charter data) that can then be used for location determination processing as described above.

For other acceptable receive processing, methods are well known to those skilled in the art for "hooking" customized processing into application processing of sought data received. For example, in an email application, a callback function API is preferably made available to the present disclosure so that every time an applicable received email distribution is received with specified criteria (e.g. certain subject, certain attached file name, certain source, or any other identifiable email attribute(s) (provided by present disclosure processing to API) sent by block **4430**, the callback function (provided by present disclosure processing to the appropriate API) is invoked for custom processing. In this example, the present disclosure invokes the callback API for providing: the callback function to be invoked, and the email criteria for triggering invocation of the callback function; for processing of permissions or charter data. For example, a unique subject field indicates to the email application that the email item should be directed by the email application to the callback function for processing. The present disclosure callback function then parses permissions and/or charter information from the email item and updates local permissions **10** and/or charters **12**. Data received in the email item may be textual syntax derived from the BNF grammar in an email body or attached file form, XML syntax derived from the BNF grammar in

email body or attached file form, an X.409 binary encoding in attached file form, or other appropriate format received with the email item (e.g. new Document Interchange Architecture (DIA) attribute data, etc). A process return status is preferably returned by the callback function, for example for appropriate email confirmation of delivery processing.

In another embodiment, the present disclosure provides at least one thread of processing for polling a known API, or email repository, for sought criteria (e.g. attributes) which identifies the email item as destined for present disclosure processing. Once the email item(s) are found, they are similarly parsed and processed for updating permissions **10** and/or charters **12**.

Thus, there are well known methods for processing data in context of this disclosure for receiving permissions **10** and/or charters **12** from an originating MS to a receiving MS, for example when using email. Similarly (callback function or polling), SMS messages can be used to communicate data **10** and/or **12** from one MS to another MS, albeit at smaller data exchange sizes. The sending MS may break up larger portions of data which can be sent as parse-able text (e.g. source syntax, XML, etc. derived from the BNF grammar) to the receiving MS. It may take multiple SMS messages to communicate the data in its entirety.

Regardless of the type of receiving application, those skilled in the art recognize many clever methods for receiving data in context of a MS application which communicates in a peer to peer fashion with another MS (e.g. callback function(s), API interfaces in an appropriate loop which can remain blocked until sought data is received for processing, polling known storage destinations of data received, or other applicable processing).

Permission data **10** and charter data **12** may be manually copied from one MS to another over any appropriate communications connection between the MSs. Permission data **10** and charter data **12** may also be manually copied from one MS to another MS using available file management system operations (move or copy file/data processing). For example, a special directory can be defined which upon deposit of a file to it, processing parses it, validates it, and uses it to update permissions **10** and/or charters **12**. Errors found may also be reported to the user, but preferably there are automated processes that create/maintain the file data to prevent errors in processing. Any of a variety of communications wave forms can be used depending on MS capability.

FIG. **45** depicts a flowchart for describing a preferred embodiment of MS charters configuration processing of block **1482**. FIG. **45** is of Self Management Processing code **18**. Processing starts at block **4502** and continues to block **4504** where a list of charters configuration options are presented to the user. Thereafter, block **4506** waits for a user action in response to options presented. Block **4506** continues to block **4508** when a user action has been detected. If block **4508** determines the user selected to configure charters data, then the user configures charters data at block **4510** (see FIG. **46**A) and processing continues back to block **4504**. If block **4508** determines the user did not select to configure charters data, then processing continues to block **4512**. If block **4512** determines the user selected to configure actions data, then the user configures actions data at block **4514** (see FIG. **47**A) and processing continues back to block **4504**. If block **4512** determines the user did not select to configure actions data, then processing continues to block **4516**. If block **4516** determines the user selected to configure parameter data, then the user configures parameter data at block **4518** (see FIG. **48**A) and processing continues back to block **4504**. If block **4516** determines the user did not select to configure parameter data, then processing continues to block **4520**. If block **4520** determines the user selected to view other's charter data, then block **4522** invokes the view other's info processing of FIG. **42** with CHARTER_INFO as a parameter (for viewing other's charter data) and processing continues back to block **4504**. If block **4520** determines the user did not select to view other's charter data, then processing continues to block **4524**. If block **4524** determines the user selected to view other's actions data, then block **4526** invokes the view other's info processing of FIG. **42** with ACTION_INFO as a parameter (for viewing other's action data) and processing continues back to block **4504**. If block **4524** determines the user did not select to view other's action data, then processing continues to block **4528**. If block **4528** determines the user selected to view other's parameter data, then block **4530** invokes the view other's info processing of FIG. **42** with PARAMETER_INFO as a parameter (for viewing other's parameter data information) and processing continues back to block **4504**. If block **4528** determines the user did not select to view other's parameter data, then processing continues to block **4532**. If block **4532** determines the user selected to send charters data, then block **4534** invokes the send data processing of FIG. **44**A with CHARTER_INFO as a parameter (for sending charters data) and processing continues back to block **4504**. If block **4532** determines the user did not select to send charters data, then processing continues to block **4536**. If block **4536** determines the user selected to configure accepting charters, then block **4538** invokes the configure acceptance processing of FIG. **43** with CHARTER_INFO as a parameter (for configuring acceptance of charters data) and processing continues back to block **4504**. If block **4536** determines the user did not select to configure accepting charters, then processing continues to block **4540**. If block **4540** determines the user selected to exit block **1482** processing, then block **4542** completes block **1482** processing. If block **4540** determines the user did not select to exit, then processing continues to block **4544** where all other user actions detected at block **4506** are appropriately handled, and processing continues back to block **4504**.

In an alternate embodiment where the MS maintains GDRs, GADRs, CDRs, ADRS, PARMDRs and GRPDRs (and their associated data records DDRs, HDRs and TDRs) at the MS where they were configured, FIG. **45** may not provide blocks **4520** through **4530**. The MS may be aware of its user charters and need not share the data (i.e. self contained). In some embodiments, options **4520** through **4530** cause access to locally maintained data for others (other users, MSs, etc) or cause remote access to data when needed (e.g. from the remote MSs). In the embodiment where no data is maintained locally for others, blocks **4532** through **4538** may not be necessary. In sum, the preferred embodiment is to locally maintain charters data for the MS user and others (e.g. MS users) which are relevant to provide the richest set of charters governing MS processing at the MS.

FIGS. **46**A through **46**B depict flowcharts for describing a preferred embodiment of MS user interface processing for charters configuration of block **4510**. With reference now to FIG. **46**A, processing starts at block **4602**, continues to block **4604** for initialization (e.g. a start using database command), and then to block **4606** where groups the user is a member of are accessed. Block **4606** retrieves all GRPDRs **3540** joined to GADRs **3520** such that the descendant type field **3520***c* and descendant ID field **3520***d* match the user information, and the ascendant type field **3520***a* is set to Group and the ascendant ID field **3520***b* matches the group ID field **3540***a*. While there may be different types of groups as defined for the BNF grammar, the GRPDR is a derivative embodiment which hap-

pens to not distinguish. Alternate embodiments may carry a group type field to select appropriate records by group type. Yet another embodiment may not have a block **4606** with processing at block **4608** for gathering data additionally by groups the user is a member of. Block **4606** continues to block **4608**.

Block **4608** accesses all CDRs (e.g. all rows from a CDR SQL table) for the user of FIG. **46**A (e.g. user information matches field **3700***b*), and for the groups the user is a member of (e.g. group information matches field **3700***b* (e.g. owner type=group, owner id=a group ID field **3540***a* from block **4606**)). The CDRs are additionally joined (e.g. SQL join) with GDRs, DDRs and TDRs (e.g. fields **3500***t*, **3600***b* and **3640***b*=Charter and by matching ID fields **3500***a*, **3600***a* and **3640***a* with field **3700***a*). Description field **3600** can provide a useful description last saved by the user for the charter entry. Block **4608** may also retrieve system predefined data records for use and/or management. Thereafter, each joined entry returned at block **4608** is associated at block **4610** with the corresponding data IDs (at least fields **3700***a*/**3500***a* and **3540***a*) for easy unique record accesses when the user acts on the data. Block **4610** also initializes a list cursor to point to the first list entry to be presented to the user. Thereafter, block **4612** sets user interface indication for where the list cursor is currently set (e.g. set to highlight the entry), and any list scrolling settings are set (the list is initially not set for being scrolled on first FIG. **46**A processing encounter to block **4612** from block **4610**). Block **4612** continues to block **4614** where the entry list is presented to the user in accordance with the list cursor and list scroll settings managed for presentation at block **4612**. Thereafter, block **4616** waits for user action to the presented list of charters data and will continue to block **4618** when a user action has been detected. Presentation of the scrollable list preferably presents in an entry format such that an entry contains fields for: DDR **3600** description; GDR owner information, grantor information and grantee information; GRPDR owner information and group name if applicable; CDR information; and TDR time spec information. Alternate embodiments will present less information, or more information (e.g. join to ADR and/or PARMDR information).

If block **4618** determines the user selected to set the list cursor to a different entry, then block **4620** sets the list cursor accordingly and processing continues back to block **4612**. Block **4612** always sets for indicating where the list cursor is currently pointed and sets for appropriately scrolling the list if necessary when subsequently presenting the list at block **4614**. If block **4618** determines the user did not select to set the list cursor, then processing continues to block **4622**. If block **4622** determines the user selected to add a charter, then block **4624** accesses a maximum number of charters allowed (perhaps multiple maximum values accessed), and block **4626** checks the maximum(s) with the number of current charters defined. There are many embodiments for what deems a maximum (for this user, for a group, for this MS, etc). If block **4626** determines a maximum number of charters allowed already exists, then block **4628** provides an error to the user and processing continues back to block **4612**. Block **4628** preferably requires the user to acknowledge the error before continuing back to block **4612**. If block **4626** determines a maximum was not exceeded, then block **4630** interfaces with the user for entering validated charter data and block **4632** adds the data record(s), appropriately updates the list with the new entry, and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **4612**. If block **4622** determines the user did not want to add a charter, processing continues to block **4634**. Block **4632** will add a CDR, GDR, DDR, HDR (to set creator information) and TDR. The DDR and TDR are optionally added by the user, but the DDR may be strongly suggested (if not enforced on the add). This will provide a charter record. Additionally, block **4630** may add new ADR(s) and/or PARMDR(s) (which are validated to exist prior to adding data at block **4632**). In one embodiment, a GDR associated to the CDR is not added; for indicating the user wants his charter made available to all other user MSs which are willing to accept it.

If block **4634** determines the user selected to delete a charter, then block **4636** deletes the data record currently pointed to by the list cursor, modifies the list for the discarded entry, and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **4612**. Block **4636** will use the Charter ID field **3700***a*/**3500***a* (associated with the entry at block **4610**) to delete the charter. Associated CDR, ADR(s), PARMDR(s), DDR **3600**, HDR **3620**, and TDR **3640** is also deleted (e.g. preferably with a cascade delete in a SQL embodiment). If block **4634** determines the user did not select to delete a charter, then processing continues to block **4652** of FIG. **46**B by way of off-page connector **4650**.

With reference now to FIG. **46**B, if block **4652** determines the user selected to modify a charter, then block **4654** interfaces with the user to modify charter data of the entry pointed to by the list cursor. The user may change information of the GDR, CDR, ADR and/or PARMDR and any associated records (e.g. DDR and TDR). The user may also add applicable records at block **4654**. Block **4654** waits for a user action indicating completion. Block **4654** will continue to block **4656** when the complete action is detected. If block **4656** determines the user exited, then processing continues back to block **4612** by way of off-page connector **4698**. If block **4656** determines the user selected to save changes made at block **4654**, then block **4658** updates the data and the list is appropriately updated before continuing back to block **4612**. Block **4658** may update the GDR, CDR, ADR, PARMDR and/or any associated records (e.g. DDR, and/or TDR) using the charter id field **3700***a*/**3500***a* (associated to the entry at block **4610**). Block **4658** will update an associated HDR as well. Block **4658** may add new CDR, ADR(s), PARMDR(s), a DDR and/or TDR as part of the charter change. If block **4652** determines the user did not select to modify a charter, then processing continues to block **4660**.

If block **4660** determines the user selected to get more details of the charter (e.g. show all joinable data to the GDR or CDR that is not already presented with the entry), then block **4662** gets additional details (may involve database queries in an SQL embodiment) for the charter pointed to by the list cursor, and block **4664** appropriately presents the information to the user. Block **4664** then waits for a user action that the user is complete reviewing details, in which case processing continues back to block **4612**. If block **4660** determines the user did not select to get more detail, then processing continues to block **4666**.

If block **4666** determines the user selected to internalize charters data thus far being maintained, then block **4668** internalizes (e.g. as a compiler would) all applicable data records for well performing use by the MS, and block **4670** saves the internalized form, for example to MS high speed non-persistent memory. In one embodiment, blocks **4668** and **4670** internalize charter data to applicable C structures of FIGS. **34**A through **34**G (also see FIG. **52**). In various embodiments, block **4668** maintains statistics for exactly what was internalized, and updates any running totals or averages maintained for a plurality of internalizations up to this point, or over certain time periods. Statistics such as:

number of active constructs; number of user construct edits of particular types; amount of associated storage used, freed, changed, etc with perhaps a graphical user interface to graph changes over time; number of charter expressions, actions, term types, etc specified, number of charters affected and unaffected by permissions; and other charter dependent statistics. In other embodiments, statistical data is initialized at internalization time to prepare for subsequent gathering of useful statistics during charter processing. In embodiments where a tense qualifier is specified for TimeSpec information, saving the internalized form at block **4670** causes all past and current tense configurations to become effective for being processed.

Block **4670** then continues back to block **4612**. If block **4666** determines the user did not select to internalize charter configurations, then processing continues to block **4672**. Alternate embodiments of processing charters **12** in the present disclosure will rely upon the data records entirely, rather than requiring the user to redundantly internalize from persistent storage to non-persistent storage for use. Persistent storage may be of reasonably fast performance to not require an internalized version of charters **12**. Different embodiments may completely overwrite the internalized form, or update the current internalized form with any changes.

If block **4672** determines the user selected to exit block **4510** processing, then block **4674** cleans up processing thus far accomplished (e.g. issue a stop using database command), and block **4676** completes block **4510** processing. If block **4672** determines the user did not select to exit, then processing continues to block **4678** where all other user actions detected at block **4616** are appropriately handled, and processing continues back to block **4616** by way off off-page connector **4696**.

FIGS. **47**A through **47**B depict flowcharts for describing a preferred embodiment of MS user interface processing for actions configuration of block **4514**. With reference now to FIG. **47**A, processing starts at block **4702**, continues to block **4704** for initialization (e.g. a start using database command), and then to block **4706** where groups the user is a member of are accessed. Block **4706** retrieves all GRPDRs **3540** joined to GADRs **3520** such that the descendant type field **3520***c* and descendant ID field **3520***d* match the user information, and the ascendant type field **3520***a* is set to Group and the ascendant ID field **3520***b* matches the group ID field **3540***a*. While there may be different types of groups as defined for the BNF grammar, the GRPDR **3540** is a derivative embodiment which happens to not distinguish. Alternate embodiments may carry a group type field to select appropriate records by group type. Yet another embodiment may not have a block **4706** with processing at block **4708** for gathering data additionally by groups the user is a member of. Block **4706** continues to block **4708**.

Block **4708** accesses all ADRs (e.g. all rows from a ADR SQL table) for the user of FIG. **47**A matching the owner information of the ADRs (e.g. user information matches field **3750***b*) to the user and to groups the user is a member of (e.g. group information matches field **3750***b* (e.g. owner type=group, owner id=group ID field **3540***a* from block **4706**). The ADRs are additionally joined (e.g. SQL join) with DDRs **3600** and TDRs **3640** (e.g. fields **3600***b* and **3640***b*=Action and by matching ID fields **3600***a* and **3640***a* with field **3750***a*). Description field **3600***c* can provide a useful description last saved by the user for the action data. Block **4708** may also retrieve system predefined data records for use and/or management. Thereafter, each joined entry returned at block **4708** is associated at block **4710** with the corresponding data IDs (at least fields **3750***a* and **3540***a*) for easy unique record accesses when the user acts on the data. Block **4710** also initializes a list cursor to point to the first action item to be presented to the user in the list. Thereafter, block **4712** sets user interface indication for where the list cursor is currently set (e.g. set to highlight the entry) and any list scrolling settings are set (the list is initially not set for being scrolled on first FIG. **47**A processing encounter to block **4712** from, block **4710**. Block **4712** continues to block **4714** where the entry list is presented to the user in accordance with the list cursor and list scroll settings managed for presentation at block **4712**. Thereafter, block **4716** waits for user action to the presented list of action data and will continue to block **4718** when a user action has been detected. Presentation of the scrollable list preferably presents in an entry format reference-able by the list cursor. An action entry presented preferably contains ADR fields including owner information; GRPDR owner information and group name if applicable; TDR time spec information; and DDR information. Alternate embodiments will present less information, or more information (e.g. join ADR(s) to PARMDR(s) via field(s) **3750***g*).

If block **4718** determines the user selected to set the list cursor to a different action entry, then block **4720** sets the list cursor accordingly and processing continues back to block **4712**. Block **4712** always sets for indicating where the list cursor is currently pointed and sets for appropriately scrolling the list if necessary when subsequently presenting the list at block **4714**. If block **4718** determines the user did not select to set the list cursor, then processing continues to block **4722**. If block **4722** determines the user selected to add an action, then block **4724** accesses a maximum number of actions allowed (perhaps multiple maximum values accessed), and block **4726** checks the maximum(s) with the number of current actions defined. There are many embodiments for what deems a maximum (for this user, for a group, for this MS, etc). If block **4726** determines a maximum number of actions allowed already exists, then block **4728** provides an error to the user and processing continues back to block **4712**. Block **4728** preferably requires the user to acknowledge the error before continuing back to block **4712**. If block **4726** determines a maximum was not exceeded, then block **4730** interfaces with the user for entering validated action data and block **4732** adds the data record, appropriately updates the list with the new entry, and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **4712**. If block **4722** determines the user did not want to add an action, processing continues to block **4734**. Block **4732** will add an ADR, HDR **3620** (to set creator information) and TDR **3640**. The DDR and TDR are optionally added by the user. Additionally, at block **4730** the user may add new PARMDR(s) for the action.

If block **4734** determines the user selected to modify an action, then block **4736** interfaces with the user to modify action data of the entry pointed to by the list cursor. The user may change information of the ADR and any associated records (e.g. DDR, TDR). The user may also add the associated records at block **4736**. Block **4736** waits for a user action indicating completion. Block **4736** will continue to block **4738** when the action is detected at block **4736**. If block **4738** determines the user exited, then processing continues back to block **4712**. If block **4738** determines the user selected to save changes made at block **4736**, then block **4740** updates the data and the list is appropriately updated before continuing back to block **4712**. Block **4740** may update the ADR and/or any associated records (e.g. DDR and/or TDR) using the action id field **3750***a* (associated to the action item at block **4710**). Block **4740** will update an associated HDR as well. Block **4736** may add a new a DDR and/or TDR as part of the action

change. If block **4734** determines the user did not select to modify an action, then processing continues to block **4752** by way of off-page connector **4750**.

With reference now to FIG. **47**B, if block **4752** determines the user selected to get more details of the action (e.g. show all joinable data to the ADR that is not already presented with the entry), then block **4754** gets additional details (may involve database queries in an SQL embodiment) for the action pointed to by the list cursor, and block **4756** appropriately presents the information to the user. Block **4756** then waits for a user action that the user is complete reviewing details, in which case processing continues back to block **4712** by way of off-page connector **4798**. If block **4752** determines the user did not select to get more detail, then processing continues to block **4758**.

If block **4758** determines the user selected to delete an action, then block **4760** determines any data records (e.g. CDR(s)) that reference the action data record to be deleted. Preferably, no referencing data records (e.g. CDRs) are joinable (e.g. field **3700***d*) to the action data record being deleted, otherwise the user may improperly delete an action from a configured charter. The user should remove ascending references to an action for deletion first. Block **4760** continues to block **4762**. If block **4762** determines there was at least one CDR reference, block **4764** provides an appropriate error with the reference(s) found so the user can subsequently reconcile. Block **4764** preferably requires the user to acknowledge the error before continuing back to block **4712**. If no references were found as determined by block **4762**, then processing continues to block **4766** for deleting the data record currently pointed to by the list cursor. Block **4766** also modifies the list for the discarded entry, and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **4712**. Block **4766** will use the action ID field **3750***a* (associated with the entry at block **4710**) to delete an action. Associated records (e.g. DDR **3600**, HDR **3620**, and TDR **3640**) are also deleted (e.g. preferably with a cascade delete in a SQL embodiment). If block **4758** determines the user did not select to delete an action, then processing continues to block **4768**.

If block **4768** determines the user selected to exit block **4514** processing, then block **4770** cleans up processing thus far accomplished (e.g. issue a stop using database command), and block **4772** completes block **4514** processing. If block **4768** determines the user did not select to exit, then processing continues to block **4774** where all other user actions detected at block **4716** are appropriately handled, and processing continues back to block **4716** by way off off-page connector **4796**.

FIGS. **48**A through **48**B depict flowcharts for describing a preferred embodiment of MS user interface processing for parameter information configuration of block **4518**. With reference now to FIG. **48**A, processing starts at block **4802**, continues to block **4804** for initialization (e.g. a start using database command), and then to block **4806** where groups the user is a member of are accessed. Block **4806** retrieves all GRPDRs **3540** joined to GADRs **3520** such that the descendant type field **3520***c* and descendant ID field **3520***d* match the user information, and the ascendant type field **3520***a* is set to Group and the ascendant ID field **3520***b* matches the group ID field **3540***a*. While there may be different types of groups as defined for the BNF grammar, the GRPDR **3540** is a derivative embodiment which happens to not distinguish. Alternate embodiments may carry a group type field to select appropriate records by group type. Yet another embodiment may not have a block **4806** with processing at block **4808** for gathering data additionally by groups the user is a member of. Block **4806** continues to block **4808**.

Block **4808** accesses all PARMDRs (e.g. all rows from a PARMDR SQL table) for the user of FIG. **48**A matching the owner information of the PARMDRs (e.g. user information matches field **3775***b*) to the user and to groups the user is a member of (e.g. group information matches field **3775***b* (e.g. owner type=group, owner id=group ID field **3540***a* from block **4806**). The PARMDRs are additionally joined (e.g. SQL join) with DDRs **3600** (e.g. field **3600***b*=Parameter and by matching ID field **3600***a* with field **3775***a*). Description field **3600***c* can provide a useful description last saved by the user for the parameter data. Block **4808** may also retrieve system predefined data records for use and/or management. Thereafter, each joined entry returned at block **4808** is associated at block **4810** with the corresponding data IDs (at least fields **3775***a* and **3540***a*) for easy unique record accesses when the user acts on the data. Block **4810** also initializes a list cursor to point to the first parameter entry to be presented to the user in the list. Thereafter, block **4812** sets user interface indication for where the list cursor is currently set (e.g. set to highlight the entry) and any list scrolling settings are set (the list is initially not set for being scrolled on first FIG. **48**A processing encounter to block **4812** from block **4810**). Block **4812** continues to block **4814** where the entry list is presented to the user in accordance with the list cursor and list scroll settings managed for presentation at block **4812**. Thereafter, block **4816** waits for user action to the presented list of parameter data and will continue to block **4818** when a user action has been detected. Presentation of the scrollable list preferably presents in an entry format reference-able by the list cursor. A parameter entry presented preferably contains fields for: PARMDR field **3775***c*; GRPDR owner information; owning GRPDR owner information and group name if applicable; and DDR information. Alternate embodiments will present less information, or more information (e.g. commands and operands parameters may be used with, parameter descriptions, etc).

If block **4818** determines the user selected to set the list cursor to a different parameter entry, then block **4820** sets the list cursor accordingly and processing continues back to block **4812**. Block **4812** always sets for indicating where the list cursor is currently pointed and sets for appropriately scrolling the list if necessary when subsequently presenting the list at block **4814**. If block **4818** determines the user did not select to set the list cursor, then processing continues to block **4822**. If block **4822** determines the user selected to add a parameter, then block **4824** accesses a maximum number of parameter entries allowed (perhaps multiple maximum values accessed), and block **4826** checks the maximum(s) with the number of current parameter entries defined. There are many embodiments for what deems a maximum (for this user, for a group, for this MS, etc). If block **4826** determines a maximum number of parameter entries allowed already exists, then block **4828** provides an error to the user and processing continues back to block **4812**. Block **4828** preferably requires the user to acknowledge the error before continuing back to block **4812**. If block **4826** determines a maximum was not exceeded, then block **4830** interfaces with the user for entering validated parameter data, and block **4832** adds the data record, appropriately updates the list with the new entry, and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **4812**. If block **4822** determines the user did not want to add a parameter entry, processing continues to block **4834**. Block

4832 will add a PARMDR, DDR 3600 and HDR 3620 (to set creator information). The DDR is optionally added by the user.

If block 4834 determines the user selected to modify a parameter entry, then block 4836 interfaces with the user to modify parameter data of the entry pointed to by the list cursor. The user may change information of the PARMDR and any associated records (e.g. DDR). The user may also add the associated records at block 4836. Block 4836 waits for a user action indicating completion. Block 4836 will continue to block 4838 when the complete action is detected at block 4836. If block 4838 determines the user exited, then processing continues back to block 4812. If block 4838 determines the user selected to save changes made at block 4836, then block 4840 updates the data and the list is appropriately updated before continuing back to block 4812. Block 4840 may update the PARMDR and/or any associated DDR using the parameter id field 3775*a* (associated to the parameter entry at block 4810). Block 4840 will update an associated HDR as well. Block 4836 may add a new DDR as part of the parameter entry change. If block 4834 determines the user did not select to modify a parameter, then processing continues to block 4852 by way of off-page connector 4850.

With reference now to FIG. 48B, if block 4852 determines the user selected to get more details of the parameter entry, then block 4854 gets additional details (may involve database queries in an SQL embodiment) for the parameter entry pointed to by the list cursor, and block 4856 appropriately presents the information to the user. Block 4856 then waits for a user action that the user is complete reviewing details, in which case processing continues back to block 4812 by way of off-page connector 4898. If block 4852 determines the user did not select to get more detail, then processing continues to block 4858.

If block 4858 determines the user selected to delete a parameter entry, then block 4860 determines any data records (e.g. ADR(s)) that reference the parameter data record to be deleted. Preferably, no referencing data records (e.g. ADRs) are joinable (e.g. field 3750*g*) to the parameter data record being deleted, otherwise the user may improperly delete a parameter from a configured action. The user should remove references to a parameter entry for deletion first. Block 4860 continues to block 4862. If block 4862 determines there was at least one reference, block 4864 provides an appropriate error with the reference(s) found so the user can subsequently reconcile. Block 4864 preferably requires the user to acknowledge the error before continuing back to block 4812. If no references were found as determined by block 4862, then processing continues to block 4866 for deleting the data record currently pointed to by the list cursor, along with any other related records that can be deleted. Block 4866 also modifies the list for the discarded entry(s), and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block 4812. Block 4866 will use the parameter ID field 3775*a* (associated with the entry at block 4810) to delete the parameter entry. Associated records (e.g. DDR 3600, and HDR 3620) are also deleted (e.g. preferably with a cascade delete in a SQL embodiment). If block 4858 determines the user did not select to delete a parameter entry, then processing continues to block 4868.

If block 4868 determines the user selected to exit block 4518 processing, then block 4870 cleans up processing thus far accomplished (e.g. issue a stop using database command), and block 4872 completes block 4518 processing. If block 4868 determines the user did not select to exit, then processing continues to block 4874 where all other user actions detected at block 4816 are appropriately handled, and processing continues back to block 4816 by way off off-page connector 4896.

FIGS. 39A, 40A, 41A, 46A, 47A and 48A assume a known identity of the user for retrieving data records. Alternate embodiments may provide a user interface option (e.g. at block 3904/4004/4104/4604/4704/4804) for whether the user wants to use his own identity, or a different identity (e.g. impersonate another user, a group, etc). In this embodiment, processing (e.g. block 3904/4004/4104/4604/4704/4804) would check permissions/privileges for the user (of FIGS. 39A, 40A, 41A, 46A, 47A and/or 48A) for whether or not an impersonation privilege was granted by the identity the user wants to act on behalf of. If no such privilege was granted, an error would be presented to the user. If an impersonation privilege was granted to the user, then applicable processing (FIGS. 39A&B, FIGS. 40A&B, FIGS. 41A&B, FIGS. 46A&B, FIGS. 47A&B and/or FIGS. 48A&B) would continue in context of the permitted impersonated identity. In another embodiment, an impersonation privilege could exist from a group to another identity for enforcing who manages grants for the group (e.g. 3904/4004/4104/4604/4704/4804 considers this privilege for which group identity data can, and cannot, be managed by the user). One privilege could govern who can manage particular record data for the group. Another privilege can manage who can be maintained to a particular group. Yet another embodiment could have a specific impersonation privilege for each of FIGS. 39A&B, FIGS. 40A&B, FIGS. 41A&B, FIGS. 46A&B, FIGS. 47A&B and/or FIGS. 48A&B. Yet another embodiment uses Grantor field information (e.g. fields 3500*c* and 3500*d*) for matching to the user's identity(s) (user and/or group(s)) for processing when the choice is available (e.g. in a GDR for permissions and/or charters).

FIGS. 39A, 40A, 41A, 46A, 47A and 48A may also utilize VDRs 3660 if referenced in any data record fields of processing for elaboration to constructs or values that are required at a processing block. Appropriate variable name referencing syntax, or variable names referenced in data record fields, will be used to access VDR information for elaboration to the value(s) that are actually needed in data record information when accessed.

FIG. 49A depicts an illustration for preferred permission data 10 processing in the present disclosure LBX architecture, for example when WDRs are in-process of being maintained to queue 22, or being inbound to a MS (referred to generally as "incoming" in FIG. 49A). Table 4920 depicts considerations for privilege data (i.e. permission data 10) resident at the MS of a first identity ID_1(grammar ID/IDType), depending on privileges granted in the following scenarios:

1) The first identity ID_1 (Grantor) granting a privilege to a second identity ID_2 (Grantee; grammar ID/IDType), as shown in cell 4924: Privilege data is maintained by ID_1 at the ID_1 MS as is used to govern actions, functionality, features, and/or behavior for the benefit of ID_2, by a) processing ID_1 WDR information at the ID_2 MS (preferably, privileges are communicated to ID_2 MS for enforcing and/or cloning there), b) processing ID_2 WDR information at the ID_1 MS (privileges locally maintained to ID_1), and c) processing ID_1 WDR information at the ID_1 MS (privileges locally maintained to ID_1);
2) The first identity ID_1 (Grantor) granting a privilege to himself (Grantee), as shown in cell 4922: Preferably, privilege data in this case is not necessary, no configuration interface is required for this scenario, and an identity implicitly has all conceivable privileges assigned to

himself by default; however, alternatively privileges may be appropriate for activating/deactivating functionality;

3) The second identity ID_2 (Grantor) granting a privilege to the first identity (Grantee), as shown in cell **4926**: Privilege data is used for informing ID_1 (or enabling ID_1 to clone per a privilege) and to govern actions, functionality, features, and/or behavior for the benefit of ID_1, by a) processing ID_2 WDR information at the ID_1 MS (preferably, privileges are communicated to ID_1 MS for enforcing and/or cloning there), b) processing ID_1 WDR information at the ID_2 MS (privileges locally maintained to ID_2); and c) processing ID_2 WDR information at the ID_2 MS (privileges locally maintained to ID_2); and/or

4) The second identity granting a privilege to himself, as shown in cell **4928**: Preferably, privilege data in this case is not necessary, no communications interface is required for this scenario, and an identity implicitly has all conceivable privileges assigned to himself by default; however, alternatively privileges may be appropriate for activating/deactivating functionality.

Table **4940** depicts considerations for privilege data (i.e. permission data **10**) resident at the MS of a second identity ID_2 (grammar ID/IDType), depending on privileges granted in the following scenarios:

5) A first identity ID_1 (Grantor) granting a privilege to the second identity ID_2 (Grantee; grammar ID/IDType), as shown in cell **4944**: Privilege data is used for informing ID_2 (or enabling ID_2 to clone per a privilege) and to govern actions, functionality, features, and/or behavior for the benefit of ID_2, by a) processing ID_1 WDR information at the ID_2 MS (preferably, privileges are communicated to ID_1 MS for enforcing and/or cloning there), b) processing ID_2 WDR information at the ID_1 MS (privileges locally maintained to ID_1), and c) processing ID_1 WDR information at the ID_1 MS (privileges locally maintained to ID_1);

6) The first identity ID_1 (Grantor) granting a privilege to himself (Grantee), as shown in cell **4942**: Preferably, privilege data in this case is not necessary, no communications interface is required for this scenario, and an identity implicitly has all conceivable privileges assigned to himself by default; however, alternatively privileges may be appropriate for activating/deactivating functionality;

7) The second identity ID_2 (Grantor) granting a privilege to the first identity (Grantee), as shown in cell **4946**: Privilege data is maintained by ID_2 at the ID_2 MS as is used to govern actions, functionality, features, and/or behavior for the benefit of ID_1, by a) processing ID_2 WDR information at the ID_1 MS (preferably, privileges are communicated to ID_1 MS for enforcing and/or cloning there), b) processing ID_1 WDR information at the ID_2 MS (privileges locally maintained to ID_2) and c) processing ID_2 WDR information at the ID_2 MS (privileges locally maintained to ID_2); and/or

8) The second identity granting a privilege to himself, as shown in cell **4948**: Preferably, privilege data in this case is not necessary, no configuration interface is required for this scenario, and an identity implicitly has all conceivable privileges assigned to himself by default; however, alternatively privileges may be appropriate for activating/deactivating functionality.

FIG. **49**B depicts an illustration for preferred charter data **12** processing in the present disclosure LBX architecture, for example when WDRs are in-process of being maintained to queue **22**, or being inbound to a MS (referred to generally as "incoming" in FIG. **49**B). Table **4960** depicts considerations for charter data resident at the MS of a first identity ID_1 (grammar ID/IDType), depending on privileges granted in the following scenarios:

1) The first identity ID_1 (Grantee) owning a charter for use at the MS of a second identity ID_2 (Grantor; grammar ID/IDType), as shown in cell **4964**: Charter data is maintained by ID_1 at the ID_1 MS for being candidate use at the ID_2 MS to cause actions, functionality, features, and/or behavior, in accordance with configured permission data **10**, for the benefit of either ID_1 or ID_2 by a) processing ID_2 WDR information at the ID_2 MS (preferably, charters are communicated to ID_2 MS for use there), and b) processing ID_1 WDR information at the ID_2 MS (preferably, charters are communicated to ID_2 MS for use there);

2) The first identity ID_1 (Grantee) owning a charter for use at his own MS, as shown in cell **4962**: Charter data is maintained locally for local use to cause actions, functionality, features, and/or behavior, in accordance with configured permission data **10**, for the benefit of either ID_1 or ID_2 by a) processing ID_1 WDR information at the ID_1 MS, and b) processing ID_2 WDR information at the ID_1 MS;

3) The second identity ID_2 (Grantee) owning a charter for use at the MS of the first identity ID_1 (Grantor; grammar ID/IDType), as shown in cell **4966**: Charter data is used at the ID_1 MS for informing ID_1 and enforcing cause of actions, functionality, features, and/or behavior, in accordance with configured permission data **10**, for the benefit of either ID_1 or ID_2 by a) processing ID_2 WDR information at the ID_1 MS (preferably, charters are communicated to ID_1 MS for use there), and b) processing ID_1 WDR information at the ID_1 MS (preferably, charters are communicated to ID_1 MS for use there); and/or

4) The second identity ID_2 (Grantee) owning a charter at his own MS, as shown in cell **4968**: Charter data may be communicated to the ID_1 MS for informing ID_1, allowing ID_1 to browse, or allowing ID_1 to use as a template for cloning and then making/maintaining into ID_1's own charter, wherein each reason for communicating to the ID_1 MS (or processing at the ID_1 MS) has a privilege grantable from ID_2 to ID_1.

Table **4980** depicts considerations for charter data resident at the MS of a second identity ID_2 (grammar ID/IDType), depending on privileges granted in the following scenarios:

5) The first identity ID_1 (Grantee) owning a charter for use at the MS of the second identity ID_2 (Grantor), as shown in cell **4984**: Charter data is used at the ID_2 MS for informing ID_2 and enforcing cause of actions, functionality, features, and/or behavior, in accordance with configured permission data **10**, for the benefit of either ID_1 or ID_2 by a) processing ID_2 WDR information at the ID_2 MS (preferably, charters are communicated to ID_2 MS for use there), and b) processing ID_1 WDR information at the ID_2 MS (preferably, charters are communicated to ID_2 MS for use there);

6) The first identity ID_1 (Grantee) owning a charter for use at his own MS, as shown in cell **4982**: Charter data may be communicated to the ID_2 MS for informing ID_2, allowing ID_2 to browse, or allowing ID_2 to use as a template for cloning and then making into ID_2's own charter, wherein each reason for communicating to the ID_2 MS (or processing at the ID_1 MS) has a privilege grantable from ID_1 to ID_2.

7) The second identity ID_2 (Grantee) owning a charter for use at the MS of the first identity ID_1 (Grantor; grammar

ID/IDType), as shown in cell **4986**: Charter data is maintained by ID_2 at the ID_2 MS for being candidate use at the ID_1 MS to cause actions, functionality, features, and/or behavior, in accordance with configured permission data **10**, for the benefit of either ID_1 or ID_2 by a) processing ID_2 WDR information at the ID_1 MS (preferably, charters are communicated to ID_1 MS for use there), and b) processing ID_1 WDR information at the ID_1 MS (preferably, charters are communicated to ID_1 MS for use there); and/or

8) The second identity ID_2 (Grantee) owning a charter at his own MS, as shown in cell **4988**: Charter data is maintained locally for local use to cause actions, functionality, features, and/or behavior, in accordance with configured permission data **10**, for the benefit of either ID_1 or ID_2 by a) processing ID_1 WDR information at the ID_2 MS, and b) processing ID_2 WDR information at the ID_2 MS.

Various embodiments will implement any reasonable subset of the considerations of FIGS. **49**A and **49**B, for example to minimize or eliminate communicating a user's permissions **10** and/or charters **12** to another MS, or to prevent storing the same permissions and/or charters data at more than one MS. FIGS. **49**A and **49**B are intended to highlight feasible embodiments wherein FIG. **49**B terminology "incoming" is used generally for referring to WDRs in-process which are a) being maintained (e.g. "incoming" as being maintained to queue **22**); and b) incoming to a particular MS (e.g. "incoming" as being communicated to the MS).

In one subset embodiment, privileges and charters are only maintained at the MS where they are configured for driving LBX features and functionality. In another embodiment, privileges are maintained at the MS where they were configured as well as any MSs which are relevant for those configurations, yet charters are only maintained at the MS where they are configured. In yet another embodiment, privileges and charters are maintained at the MS where they were configured, as well as any MSs which are relevant for those configurations. In another embodiment, a MS may not have all privileges assigned to itself (said to be assigned to the user of the MS) by default. Privileges may require being enabled as needed for any users to have the benefits of the associated LBX features and functionality. Thus, the considerations highlighted by FIGS. **49**A and **49**B are to "cover many bases" with any subset embodiment within the scope of the present disclosure.

Preferably, statistics are maintained by WITS for counting occurrences of each variety of the FIGS. **49**A and **49**B processing scenarios. WITS processing should also keep statistics for the count by privilege, and by charter, of each applicable WITS processing event which was affected. Other embodiments will maintain more detailed statistics by MS ID, Group ID, or other "labels" for categories of statistics. Still other embodiments will categorize and maintain statistics by locations, time, applications in use at time of processing scenarios, etc. Applicable statistical data can be initialized at internalization time to prepare for proper gathering of useful statistics during WITS processing.

FIGS. **50**A through **50**C depict an illustration of data processing system wireless data transmissions over some wave spectrum for further explaining FIGS. **13**A through **13**C, respectively. Discussions above for FIGS. **13**A through **13**C are expanded in explanation for FIGS. **50**A through **50**C, respectively. It is well understood that the DLM **200***a* (FIGS. **13**A and **50**A), ILM **1000***k* (FIGS. **13**B and **50**B) and service(s) (FIGS. **13**C and **50**C) can be capable of communicating bidirectionally. Nevertheless, FIGS. **50**A through **50**C clarify FIGS. **13**A through **13**C, respectively, with a bidirectional arrow showing data flow "in the vicinity" of the DLM **200***a*, ILM **1000***k*, and service(s), respectively. All disclosed descriptions for FIGS. **13**A through **13**C are further described by FIGS. **50**A through **50**C, respectively.

With reference now to FIG. **50**A, "in the vicinity" language is described in more detail for the MS (e.g. DLM **200***a*) as determined by clarified maximum range of transmission **1306**. In some embodiments, maximum wireless communications range (e.g. **1306**) is used to determine what is in the vicinity of the DLM **200***a*. In other embodiments, a data processing system **5090** may be communicated to as an intermediary point between the DLM **200***a* and another data processing system **5000** (e.g. MS or service) for increasing the distance of "in the vicinity" between the data processing systems to carry out LBX peer to peer data communications. Data processing system **5090** may further be connected to another data processing system **5092**, by way of a connection **5094**, which is in turn connected to a data processing system **5000** by wireless connectivity as disclosed. Data processing systems **5090** and **5092** may be a MS, service, router, switch, bridge, or any other intermediary data processing system (between peer to peer interoperating data processing systems **200***a* and **5000**) capable of communicating data with another data processing system. Connection **5094** may be of any type of communications connection, for example any of those connectivity methods, options and/or systems discussed for FIG. **1**E. Connection **5094** may involve other data processing systems (not shown) for enabling peer to peer communications between DLM **200***a* and data processing system **5000**. FIG. **50**A clarifies that "in the vicinity" is conceivably any distance from the DLM **200***a* as accomplished with communications well known to those skilled in the art demonstrated in FIG. **50**A. In some embodiments, data processing system **5000** may be connected at some time with a physically connected method to data processing system **5092**, or DLM **200***a* may be connected at some time with a physically connected method to data processing system **5090**, or DLM **200***a* and data processing system **5000** may be connected to the same intermediary data processing system. Regardless of the many embodiments for DML **200***a* to communicate in a LBX peer to peer manner with data processing system **5000**, DLM **200***a* and data processing system **5000** preferably interoperate in context of the LBX peer to peer architecture. In some embodiments, data processing systems between DLM **200***a* and the data processing **5000** intercept data for tracking, book-keeping, statistics, and for maintaining data potentially accessed by service informant code **28**, however, the LBX peer to peer model is preferably not interfered with.

Data processing system **5000** may be a DLM, ILM, or service being communicated with by DML **200***a* as disclosed in the present disclosure for FIGS. **13**A through **13**C, or for FIGS. **50**A through **50**C. LBX architecture is founded on peer to peer interaction between MSs without requiring a service to middleman data, however data processing systems **5090**, **5092** and those applicable to connection **5094** can facilitate the peer to peer interactions. In some embodiments, data processing systems between DLM **200***a* and the data processing **5000** intercept data for tracking, book-keeping, statistics, and for maintaining data potentially accessed by service informant code **28**, however, the LBX peer to peer model is preferably not interfered with. Data processing system **5000** generically represents a DLM, ILM or service(s) for analogous FIGS. **13**A through **13**C processing for sending/broadcasting data such as a data packet **5002** (like **1302**/**1312**). When a Communications Key (CK) **5004** (like **1304**/**1314**) is embedded within data **5002**, data **5002** is considered usual

communications data (e.g. protocol, voice, or any other data over conventional forward channel, reverse channel, voice data channel, data transmission channel, or any other appropriate channel) which has been altered to contain CK **5004**. Data **5002** contains a CK **5004** which can be detected, parsed, and processed when received by an MS or other data processing system in the vicinity (conceivably any distance depending on embodiment) of data processing system **5000** as determined by the maximum range of transmission **5006** (like **1306**/**1316**). CK **5004** permits "piggy-backing" on current transmissions to accomplish new functionality as disclosed herein. Transmissions radiate out in all directions in a manner consistent with the wave spectrum used, and data carried thereon may or may not be encrypted (e.g. encrypted WDR information). The radius **5008** (like **1308**/**1318**) represents a first range of signal reception from data processing system **5000** (e.g. antenna thereof, perhaps by a MS. The radius **5010** (like **1310**/**1320**) represents a second range of signal reception from data processing system **5000** (e.g. antenna thereof), perhaps by a MS. The radius **5011** (like **1311**/**1322**) represents a third range of signal reception from data processing system **5000** (e.g. antenna thereof), perhaps by a MS. The radius **5006** (like **1306**/**1316**) represents a last and maximum range of signal reception from data processing system **5000** (e.g. antenna thereof), perhaps by a MS (not shown). The time of transmission from data processing system **5000** to radius **5008** is less than times of transmission from service to radiuses **5010**, **5011**, or **5006**. The time of transmission from data processing system **5000** to radius **5010** is less than times of transmission to radiuses **5011** or **5006**. The time of transmission from data processing system **5000** to radius **5011** is less than time of transmission to radius **5006**. In another embodiment, data **5002** contains a Communications Key (CK) **5004** because data **5002** is new transmitted data in accordance with the present disclosure. Data **5002** purpose is for carrying CK **5004** information for being detected, parsed, and processed when received by another MS or data processing system in the vicinity (conceivably any distance depending on embodiment) of data processing system **5000** as determined by the maximum range of transmission.

With reference now to FIG. **50**B, "in the vicinity" language is described in more detail for the MS (e.g. ILM **1000***k*) as determined by clarified maximum range of transmission **1306**. In some embodiments, maximum wireless communications range (e.g. **1306**) is used to determine what is in the vicinity of the ILM **1000***k*. In other embodiments, a data processing system **5090** may be communicated to as an intermediary point between the ILM **1000***k* and another data processing system **5000** (e.g. MS or service) for increasing the distance of "in the vicinity" between the data processing systems to carry out LBX peer to peer data communications. Data processing system **5090** may further be connected to another data processing system **5092**, by way of a connection **5094**, which is in turn connected to a data processing system **5000** by wireless connectivity as disclosed. Data processing systems **5090** and **5092** may be a MS, service, router, switch, bridge, or any other intermediary data processing system (between peer to peer interoperating data processing systems **1000***k* and **5000**) capable of communicating data with another data processing system. Connection **5094** may be of any type of communications connection, for example any of those connectivity methods, options and/or systems discussed for FIG. **1**E. Connection **5094** may involve other data processing systems (not shown) for enabling peer to peer communications between ILM **1000***k* and data processing system **5000**. FIG. **50**B clarifies that "in the vicinity" is conceivably any distance from the ILM **1000***k* as accomplished with communications well known to those skilled in the art demonstrated in FIG. **50**B. In some embodiments, data processing system **5000** may be connected at some time with a physically connected method to data processing system **5092**, or ILM **1000***k* may be connected at some time with a physically connected method to data processing system **5090**, or ILM **1000***k* and data processing system **5000** may be connected to the same intermediary data processing system. Regardless of the many embodiments for ILM **1000***k* to communicate in a LBX peer to peer manner with data processing system **5000**, ILM **1000***k* and data processing system **5000** preferably interoperate in context of the LBX peer to peer architecture. In some embodiments, data processing systems between ILM **1000***k* and the data processing **5000** intercept data for tracking, book-keeping, statistics, and for maintaining data potentially accessed by service informant code **28**, however, the LBX peer to peer model is preferably not interfered with.

With reference now to FIG. **50**C, "in the vicinity" language is described in more detail for service(s) as determined by clarified maximum range of transmission **1316**. In some embodiments, maximum wireless communications range (e.g. **1316**) is used to determine what is in the vicinity of the service(s). In other embodiments, a data processing system **5090** may be communicated to as an intermediary point between the service(s) and another data processing system **5000** (e.g. MS) for increasing the distance of "in the vicinity" between the data processing systems to carry out LBX peer to peer data communications. Data processing system **5090** may further be connected to another data processing system **5092**, by way of a connection **5094**, which is in turn connected to a data processing system **5000** by wireless connectivity as disclosed. Data processing systems **5090** and **5092** may be a MS, service, router, switch, bridge, or any other intermediary data processing system (between peer to peer interoperating data processing system service(s) and **5000**) capable of communicating data with another data processing system. Connection **5094** may be of any type of communications connection, for example any of those connectivity methods, options and/or systems discussed for FIG. **1**E. Connection **5094** may involve other data processing systems (not shown) for enabling peer to peer communications between service(s) and data processing system **5000**. FIG. **50**C clarifies that "in the vicinity" is conceivably any distance from the service(s) as accomplished with communications well known to those skilled in the art demonstrated in FIG. **50**C. In some embodiments, data processing system **5000** may be connected at some time with a physically connected method to data processing system **5092**, or service(s) may be connected at some time with a physically connected method to data processing system **5090**, or service(s) and data processing system **5000** may be connected to the same intermediary data processing system. Regardless of the many embodiments for service(s) to communicate in a LBX peer to peer manner with data processing system **5000**, service(s) and data processing system **5000** preferably interoperate in context of the LBX peer to peer architecture. In some embodiments, data processing systems between service(s) and the data processing **5000** intercept data for tracking, book-keeping, statistics, and for maintaining data potentially accessed by service informant code **28**, however, the LBX peer to peer model is preferably not interfered with.

In an LN-expanse, it is important to know whether or not WDR information is of value for locating the receiving MS, for example to grow an LN-expanse with newly located MSs. FIGS. **50**A through **50**C demonstrate that WDR information sources may be great distances (over a variety of communications paths) from a particular MS receiving the WDR infor-

mation. Carrying intermediary system indication is well known in the art, for example to know the number of hops of a communications path. The preferred embodiment uses communications reference field **1100***g* to maintain whether or not the WDR encountered any intermediate systems, for example as identified with hops, network address change(s), channel extender transmission indications, or any pertinent data to indicate whether the WDR encountered anything other than a wireless transmission (e.g. directly between the sending MS and receiving MS). This provides FIG. **26**B with a means to qualify the peek at block **2634** for only those WDRs which show field **1100***g* to be over a single wireless connection from the source to the MS (i.e. block **2634** to read as "Peek all WDRS from queue **22** for confidence>confidence floor and most recent in trailing f(WTV) period of time and field **1100***g* indicating a wireless connected source over no intermediary systems"). Field **1100***g* would be set intelligently for all WDRs received and processed by the MS (e.g. inserted to queue **22**). In another embodiment, fields **1100***e* and **1100***f* are used to indicate that the WDR can be relied upon for triangulating a new location of the MS (e.g. block **2660** altered to get the next WDR from the REMOTE_MS list which did not arrive except through a single wireless path). In other embodiments, the correlation (e.g. field **1100***m*) can be used to know whether it involved more than a single wireless communications path. The requirement is to be able to distinguish between WDRs that can contribute to locating a MS and WDRs which should not be used to locate the MS. In any case, WDRs are always useful for peer to peer interactions as governed by privileges and charters (see WITS filtering discussed below).

In other embodiments, the WDR fields **1100***e* and **1100***f* information is altered to additionally contain the directly connected system whereabouts (e.g. intermediary system **5090** whereabouts) so that the MS (e.g. **1000***k*) can use that WDR information relevant for locating itself (e.g. triangulating the MS whereabouts). This ensures that a MS receives all relevant WDRs from peers and also uses the appropriate WDR information for determining its own location. FIG. **26**B would distinguish between the data that describes the remote MS whereabouts from the data useful for locating the receiving MS. A preferred embodiment always sets an indicator to at least field **1100***e*, **1100***f*, or **1100***g* for indicating that the WDR was in transit through one or more intermediary system(s). This provides the receiving MS with the ability to know whether or not the WDR was received directly from a wireless in-range MS versus a MS which can be communicated with so that the receiving MS can judiciously process the WDR information (see WITS filtering discussed below).

An alternate embodiment supports WDR information source systems which are not in wireless range for contributing to location determination of a MS. For example, a system can transmit WDR information outbound in anticipation of when it will be received by a MS, given knowledge of the communication architecture. Outbound date/time information is strategically set along with other WDR information to facilitate making a useful measurement at a receiving MS (e.g. TDOA). The only requirement is the WDR conform to a MS interface and be "true" to how fields are set for LBX interpretation and appropriate processing, for example to emulate a MS transmitting useful WDR information.

WITS filtering provides a method for filtering out (or in) WDRs which may be of use for locating the receiving MS, or are of use for permission and/or charter processing. Supporting ranges beyond a range within wireless range to a MS can cause a massive number of WDRs to be visible at a MS. Thus, only those WDRs which are of value, or are candidate for triggering permissions or charter processing, are to be processed. WITS filtering can use the source information (e.g. MS ID) or any other WDR fields, or any combination of WDR fields to make a determination if the WDR deserves further processing. The longer range embodiment of FIGS. **50**A through **50**C preferably incorporates a send transmission for directing the WDRs to MSs which have candidate privileges and/or charters in place, rather than a broadcast for communicating WDRs. Broadcasting can flood a network and may inundate MSs with information for WITS filtering, however the multithreaded LBX architecture may process efficiently even for broadcast data.

In another embodiment, a configuration can be made (user or system) wherein FIGS. **13**A through **13**C are applicable, and non-wireless range originated WDRs are always ignored. For example, a WDR Range Configuration (WRC) indicates how to perform WITS filter processing:

1) Ignore WDRs which are originated from a wirelessly connected source (e.g. within range **1306**);
2) Consider all WDRs regardless of source;
3) Ignore all WDRs regardless of source; and/or
4) Ignore WDRs which are not originated from a wirelessly connected source.

WDR fields, as described above, are to contain where the WDR originated and any relevant path it took to arrive. Block **1496** may be modified to include new blocks **1496***a*, **1496***b*, and **1496***c* such that:

- Block **1496***a* checks to see if the user selected to configure the WRC—an option for configuration at block **1406** wherein the user action to configure it is detected at block **1408**;
- Block **1496***b* is processed if block **1496***a* determines the user did select to configure the WRC. Block **1496***b* interfaces with the user for a WRC setting (e.g. a block **1496***b*–1 to prepare parameters for FIG. **18** processing, and a block **1496***b*–2 for invoking the Configure value procedure of FIG. **18** to set the WRC). Processing then continues to block **1496***c*.
- Block **1496***c* is processed if block **1496***a* determines the user did not select to configure the WRC, or as the result of processing leaving block **1496***b*. Block **1496***c* handles other user interface actions leaving block **1408** (e.g. becomes the "catch all" as currently shown in block **1496** of FIG. **14**B).

The WRC is then used appropriately by WITS processing for deciding what to do with the WDR in process. Assuming the WDR is to be processed further, and the WDR is not of use to locate the receiving MS, then permissions **10** and charters **12** are still checked for relevance of processing the WDR (e.g. MS ID matches active configurations, WDR contains potentially useful information for configurations currently in effect, etc). In an alternative embodiment, WITS filtering is performed at existing permission and charter processing blocks so as to avoid redundantly checking permissions and charters for relevance.

FIG. **51**A depicts an example of a source code syntactical encoding embodiment of permissions, derived from the grammar of FIGS. **30**A through **30**E, for example as user specified, system maintained, system communicated, system generated, etc. In one embodiment, a user may specify the source code as a portion of a hosting programming source code like C, C++, C#, Java, or any other programming language. The hosting programming source code compiler or interpreter shall recognize keywords (e.g. Permissions) to then uniquely parse and process the source code stream between associated delimiters (e.g. { . . . }) in a unique way, for example as handled by new compiler/interpreter code, or

with a processing plug-in appropriately invoked by the compiler/interpreter. This allows adapting an existing programming environment to handle the present disclosure with specific processing for the recognized source code section(s). In another embodiment, the present disclosure source code is handled as any other source code of the hosting programming environment through closely adapting the hosting programming source code syntax, incorporating new keywords and contextual processing, and maintaining data and variables like other hosting programming environment variables.

FIG. **51**A shows that a Permissions block contains "stuff" between delimiters ({,}) like C, C++, C#, and the Java programming languages (all referred hereinafter as Popular Programming Languages (PPLs)), except the reserved keyword "Permissions" qualifies the block which follows. Statements within the block are also aligned with syntax of PPLs. Here is an in-context description of FIG. **51**A:

Text(str)="Test Case #106729 (context)";

The str variable is of type Text (i.e. BNF Grammar "text string") and is set with string "Test Case #106729 (context)". Below will demonstrate variable string substitution for the substring "context" when str is instantiated.

Generic(assignPrivs)="G=Family,Work,\vuloc
[T=>20080402000130.24,<20080428; D=*str; H;]";

The assignPrivs variable is of type Generic and is set with a long string containing lots of stuff. Generic tells the internalizer to treat the assigned value as text string without any variable type validation at this time. The BNF grammar showed that variables have a type to facilitate validation at parse time of what has been assigned, however type checking is really not necessary since validation will occur in contexts when a variable is instantiated anyway. Another variable type (VarType) to introduce to the BNF grammar is "Generic" wherein anything assigned to the variable is to have its type delayed until after instantiation (i.e. when referenced later). Note that the str variable is not instantiated at this time (i.e. =the preferred embodiment, however an alternate embodiment would instantiate str at this time). Below will demonstrate a Generic variable instantiation.

```
Groups {
  LBXPHONE_USERS = Austin, Davood, Jane, Kris, Mark, Ravi, Sam,
  Tim;
  "SW Components" = "SM 1.0", "PIP 1.0", "PIPGUI 1.0", "SMGUI 1.0",
                    "COMM 1.0", "KERNEL 1.1";
}
```

Two (2) groups are defined. In this example embodiment, "Groups" is a reserved keyword identifying a groups definition block just as "Permissions" did the overall block. The "LBXPHONE_USERS" group is set to a simplified embodiment of MS IDs Austin, Davood, etc; and the "SW Components" group is set to LBX Phone software modules with current version numbers. Any specification of the BNF Grammar (e.g. group name, group member, etc) with intervening blanks can be delimited with double quotes to make blanks significant.

```
Grants // Can define Grant structure(s) prior to assignment {
...
}
```

In this example embodiment, "Grants" is a reserved keyword identifying a Grants definition block just as "Permissions" did the overall block. Statements within the Grants block are for defining Grants which may be used later for assigning privileges. "//" starts a comment line like PPLs, and "/*" . . . "*/" delimits comment lines like PPLs.

Family=\lbxall[R=0xFFFFFFFF;] [D=*str (context="Family")];

A grant named "Family" is assigned the privilege "\lbxall" and is relevant for all MS types (i.e. 0xFFFFFFFF such that the "R" is a specification for MSRelevance). \lbxall is the all inclusive privilege for all LBX privileges. \lbxall maps to a unique privilege id (e.g. maintained to field **3530***a*, FIGS. **34**F and **52** "unsigned long priv", etc). Optional specifications are made with delimiters "[" and "]", which coincidentally were used in defining the BNF grammar optional specifications. Each optional specification can have its own delimiters, or all optional specifications could have been made in a single pair of delimiters. The "D" specification is a Description specification which is set to an instantiation of the str variable using a string substitution. Thus, the Description is set to the string "Test Case #106729 (Family)".

```
Work =
[T=YYYYMMDD08:YYYYMMDD17;D=*str(context="Work");H;] {
...
};
```

A grant named "Work" is assigned as a parent grant to other grant definitions, in which case a delimited block for further grant definitions can be assigned. Optional specifications can be made for the Work grant prior to defining subordinate grants either before the Work grant block, or after the block just prior to the block terminating semicolon (";"). The Work grant has been assigned an optional "T" specification for a TimeSpec qualifying the grant to be in effect for every day of every month of every year for only the times of 8 AM through 5 PM. The Work grant also defined a Description of "Test Case #106729 (Work)". The "H" specification tells the internalizer to generate History information (e.g. FIGS. **36**B, **33**A, **34**E HISTORY, etc) for the Work grant.

"Department **232**"=\geoar,\geode,\nearar,\nearde;

The grant "Department **232**" is subordinate to "Work" and has four (4) privileges assigned, and no optional specifications.

```
"Department 458" = [D="Davood Iyadi's mgt scope";] {
    "Server Development Team" = ;
    "IbxPhone Development Team" =
        {
        "Comm Layer Guys" = \mssys;\msbios;
        "GUI girls" = \msguiload;
        "Mark and Tim" = \msapps;
    };
};
```

The grant "Department **458**" is subordinate to "Work", has an optional Description specification, and has two (2) subordinate grants defined. The grant "Server Development Team" is defined, but has no privileges or optional specifications. The grant "IbxPhone Development Team" is subordinate to "Work", has no optional specifications, and has three (3) subordinate grants defined. The grant "Comm Layer Guys" has two (2) privileges assigned (\mssys and \msbios), the grant "GUI girls" has one (1) privilege assigned (\msguiload), and the grant "Mark and Tim" has one (1) privilege assigned (\msapps).

"Accounting Department" [H;]=\track;
The grant "Accounting Department" is subordinate to "Work", has optional History information to be generated, and has one (1) privilege assigned.
Parents={Mom=\lbxall; Dad=\lbxall;};
Michael-Friends=\geoarr;\geode;
Jason-Friends=\nearar;\nearde;
The grant "Parents" is independent of the Work grant (a peer), has two (2) subordinate grants "Mom" and "Dad", each with a single privilege assigned. The grants "Michael-Friends" and "Jason-Friends" are each independent of other grants, and each have two (2) privileges assigned. A nested tree structure of Grants so far compiled which can be used for privilege assignments are:

```
Family
Work
    Department 232
    Department 458
        Server Development Team
        lbxPhone Development Team
            Comm Layer Guys
            GUI girls
            Mark and Tim
    Accounting Department
Parents
    Mom
    Dad
Michael-Friends
Jason-Friends
```

The nested structure of the source code was intended to highlight the relationship of grants defined. Note that assigning the Work grant from one ID to another ID results in assigning all privileges of all subordinate grants (i.e. \geoar;\geode; \nearar;\nearde;\mssys;\msbios;\msguiload;\msapps;\track).
Bill: LBXPHONE_USERS [G=\caller;\callee;\trkall;];
The MS ID Bill assigns (i.e. Grant specification "G") three (3) privileges to the LBXPHONE_USERS group (i.e. to each member of the group). Privileges and/or grants can be granted. The \caller privilege enables LBXPHONE_USERS member MSs to be able to call the Bill MS. The \callee privilege enables the Bill MS to call LBXPHONE_USERS member MSs. The \trkall privilege enables LBXPHONE_USERS members to use the MS local tracking application for reporting mobile whereabouts of the Bill MS. The grants are optional (i.e. "[" and "]") because without specific grants and/or privileges specified, all privileges are granted.
LBXPHONE_USERS: Bill [G=\callee;\caller;];
Each member of the LBXPHONE_USERS group assigns (i.e. Grant specification "G") two (2) privileges to the Bill MS. The \caller privilege enables the Bill MS to be able to call any of the members of the LBXPHONE_USERS group. The \callee privilege enables the LBXPHONE_USERS member MSs to call the Bill MS.
Bill:Sophia;
All system privileges are assigned from Bill to Sophia.
Bill:Brian [*assignPrivs];
The assignPrivs variable is instantiated to "G=Family,Work, \vuloc [T=>20080402000130.24,<20080428; D=*str; H;]" as though that configuration were made literally as:
Bill:Brian [G=Family,Work,\vuloc [T=>20080402000130.24,<20080428; D="Test Case #106729 (context)"; H;]];
Note the str variable is now instantiated as well. Bill grants Brian all privileges defined in the Family grant, all privileges of the Work grant, and the specific \vuloc privilege. The privilege \vuloc has optional specifications for TimeSpec (i.e. after 1 minute 30.24 seconds into Apr. 2, 2008 and prior to Apr. 28, 2008), Description, and History to be generated. The optional specifications ([. . .]) would have to be outside of the other optional delimiter specifications (e.g. [G= . . .] [. . .]) to be specifications for the Permission.
Bill:George [G=\geoall,\nearall;];
Bill assigns two (2) privileges to George.
Michael: Bill [G=Parents,Michael-Friends;];
Michael assigns to Bill the privileges \lbxall, \geoarr and \geode.
Jason: Bill [G=Parents,Jason-Friends;];
Jason assigns to Bill the privileges \lbxall, \nearar and \nearde.

FIG. **51**B depicts an example of a source code syntactical encoding embodiment of charters, derived from the grammar of FIGS. **30**A through **30**E, for example as user specified, system maintained, system communicated, system generated, etc. In one embodiment, a user may specify the source code as a portion of a hosting programming source code like C, C++, C#, Java, or any other programming language. The hosting programming source code compiler or interpreter shall recognize keywords (e.g. Charters) to then uniquely parse and process the source code stream between associated delimiters (e.g. { . . . }) in a unique way, for example as handled by new internalization (e.g. compiler/interpreter) code, or with a processing plug-in appropriately invoked by the internalizer. This allows adapting an existing programming environment to handle the present disclosure with specific processing for the recognized source code section(s). In another embodiment, the present disclosure source code is handled as any other source code of the hosting programming environment through closely adapting the hosting programming source code syntax, incorporating new keywords and contextual processing, and maintaining data and variables like other hosting programming environment variables.

It is important to understand that WDRs in process (e.g. to queue **22** (_ref), outbound (_O_ref), and inbound (_I_ref)) cause the recognized trigger of WDR processing to scan charters for testing expressions, and then performing actions for those expressions which evaluate to true. Expressions are evaluated within the context of applicable privileges. Actions are performed within the context of privileges. Thus, WDRs in process are the triggering objects for consulting charters at run time. Depending on the MS hardware and how many privileged MSs are "in the vicinity", there may be many (e.g. dozens) of WDRs in process every second at a MS. Each WDR in process at a MS is preferably in its own thread of processing (preferred architecture **1900**) so that every WDR in process has an opportunity to scan charters for conditional actions.

FIG. **51**B shows that a Charters block contains "stuff" between delimiters ({,}) like PPLs, except the reserved keyword "Charters" qualifies the block which follows. Statements within the block are also aligned with syntax of PPLs. Here is an in-context description of FIG. **51**B:
Condition(cond1)="(_location @@ \loc_my) [D="Test Case #104223 (v)";]";
The variable cond1 is of type Condition and is set accordingly. Validation of the variable type can occur here since the type is known. Cond1 is a Condition specification with an optional specification for the Description. Since the type "Generic" can be used, it may convenient to always use that.
"ms group"={"Jane", "George", "Sally"};
This is another method for specifying a group without a Groups block. The internalizer preferably treats an assignment using block delimiters outside of any special block

definitions as a group declaration. While there has been no group hierarchies demonstrated, groups within groups can certainly be accomplished like Grants.

(((_msid="Michael") & *cond1(v='Michael'))|
((_msid="Jason") & *cond1(v='Jason'))):
Invoke App myscript.cmd ("S"), Notify Autodial 214-405-6733;

_msid is a WDRTerm indicating to check the condition of the WDRs maintained to the local MS (e.g. processed for inserting to queue **22**). The condition _msid="Michael" tests if the WDR in process has a WDR MS ID field **1100***a* equal to the MS ID Michael. "&" is a CondOp. After instantiation of cond1 with the string substitution the second condition is "(_location @@ \loc_my) [D="Test Case #104223 (v)";]" which tests the WDR in process (e.g. for insertion to queue **22**) for a WDR location field **1100***c* which was at my current location (\loc_my is a system defined atomic term for "my current location" (i.e. the current location of the MS checking the WDR in process)). @@ is an atomic operator for "was at". There is an optional description specified for the condition to be generated. The expression formed on the left hand side of the colon (:) not only tests for Michael WDR information, but also Jason WDR information with the same WDR field tests. If the WDR in process (contains a MS ID=Michael AND Michael's location was at my current location at some time in the past), OR (i.e. |CondOp) the WDR in process (contains a MS ID=Jason AND Jason's location was at my current location at some time in the past), then the Actions construct (i.e. right hand side of colon) is acted upon. The "was at" atomic operator preferably causes access to LBX History **30** after a fruitless access to queue **22**. It may have been better to specify another condition for Michael and Jason WDRs to narrow the search, otherwise if LBX history is not well pruned the search may be timely. For example, the variable may have been better defined prior to use as:

Condition(cond1)="(_location (2W)$(10F)\loc_my) [D="Test Case #104223 (v)";]";

for recently in vicinity (i.e. within 10 feet) of my location in last 2 weeks helps narrow the search.

Parenthesis are used to affect how to evaluate the expression as is customary for an arithmetic expression, and can be used to determine which construct the optional specifications are for. Of course, a suitable precedence of operators is implemented. So, if the Expression evaluates to true, the actions shall be processed. There can be one or more actions processed. The first action performs an Invoke command with an Application operand and provides the parameter of "myscript.cmd("S")" which happens to be an executable script invocable on the particular MS. A parameter of "S" is passed to the script. The script can perform anything supported in the processable script at the particular MS. The second action performs a Notify command with an Autodial operand and provides the parameter of "214-405-6733". Notify Autodial will automatically perform a call to the phone number 214-405-6733 from the MS. So, if the MS of this configuration is currently at a location where Jason or Michael (in the vicinity) had been at some time before (as maintained in LBX History if necessary, or in last 2 weeks in refined example), then the two actions are processed. LBX History **30** will be searched for previous WDR information saved for Michael and Jason to see if the expression evaluates to true when queue **22** does not contain a matching WDR for Michael or Jason.

It is interesting to note that the condition "((\locByID_Michael @@ \loc_my)|(\locByID_Jason @@ \loc_my))" accomplishes the same expression shown in FIG. **51**B described above. \locRef_is an atomic term for the WDR location field with the suffix (Ref) referring to the value for test. \loc"R e f" is an acceptable format when there are significant blanks in the suffix for testing against the value of the WDR field. It is also interesting to note that the expression "(\loc_my @@ \locByID_Michael)" is quite different. The expression "(\loc_my @@ \locByID_Michael)" tests if my current location was at Michael's location in history, again checking LBX history. However, the WDR in process only provided the trigger to check permissions and charters. There is no field of the in process WDR accessed here.

((_I_msid="Brian") & (_I_location @ \loc_my) [D="multi-cond text";H;]):
Invoke App (myscript.cmd ("B")) [T=20080302;],
Notify Autodial (214-405-5422);

_I_msid is a WDRTerm indicating to check the condition of the WDRs inbound to the local MS (e.g. deposited to receive queue **26**). The condition _I_msid="Brian" tests if the inbound WDR has a WDR MS ID field **1100***a* equal to the MS ID Brian. "=" is an atomic operator. & is a CondOp. _I_location is the contents of the inbound WDR location field **1100***c*, so that the condition of (_I_location @ \loc_my) tests the inbound WDR for a WDR location field **1100***c* which is at my current location. @ is an atomic operator for "is at". There is an optional description specified for the condition as well as history information to be generated. The expression formed on the left hand side of the colon (:) tests for inbound WDRs from Brian wherein Brian is at my (i.e. receiving MS) current location. Assuming the expression evaluates to true, then the two (2) actions are performed. The actions are similar to the previous example, except the syntax is demonstrated to show parentheses may or may not be used for command/operand parameters. Also, the first action has an optional TimeSpec specification which mandates that the action only be performed any time during the day of Mar. 2, 2008. Otherwise, the first action will not be performed. The second action is always performed.

The _I_fldname syntax is a WDRTerm for inbound WDRs which makes sense for our expression above. A careless programmer/user could in fact create expressions that may never occur. For example, if the user specified _O_ instead of $_{13}$ I_, then outbound rather than inbound WDRs would be tested. ((_O_msid="Brian") & (_O_location @ \loc_my)) causes outbound WDRs to be tested (e.g. deposited to send queue **24**) for MS ID=Brian which are at my current location (i.e. current location of the MS with the configuration being discussed). Mixing _, _I_, and _O_ prefixes has certain semantic implications and must be well thought out by the user prior to making such a configuration. The charter expression is considered upon an event involving each single WDR and is preferably not used to compare to a plurality of potentially ambiguous/unrelated WDRs at the same time. A single WDR can be both in process locally (e.g. inserted to queue **22**) and inbound to the MS when received from MSs in the vicinity. It will not be known that the WDR meets both criteria until after it has been inbound and is then being inserted to queue **22**. Likewise, a single WDR can be both in process locally (e.g. inserted to queue **22**) and outbound from the MS. It will not be known that the WDR meets both criteria until after it has been retrieved from queue **22** and then ready for being sent outbound. The programmer/user can create bad configurations when mixing these syntaxes. It is therefore recommended, but not required, that users not mix WDR trigger syntax. Knowing a WDR is inbound and then in process to queue **22** is straightforward (e.g. origination other than "this MS"). Knowing a WDR was on queue **22** and is outbound is also straightforward (e.g. origination at outbound="this MS").

However, a preferred embodiment prevents mixing these syntaxes for triggered processing.

(M_sender=~emailAddrVar [T=<YYYYMMDD18]):

Notify Indicator (M_sender, \thisms) [D="Test Case #104223"; H;];

M_sender is an AppTerm for the registered Mail application (see FIGS. **53** and **55**), specifically the source address of the last email object received. ~emailAddrVar references a programmatic variable of the hosting programming environment (PPLs), namely a string variable to compare against the source address (e.g. billj@iswtechnologies.com). If the variable type does not match the AppTerm type, then the internalizer (e.g. compiler/interpreter) should flag it prior to conversion to an internalized form. Alternate embodiments will rely on run time for error handling. The Condition also specifies an optional TimeSpec specification wherein the condition for testing is only active during all seconds of the hour of 6:00 PM every day (just to explain the example). Expressions can contain both AppTerms and WDRTerms while keeping in mind that WDRs in process are the triggers for checking charters. M_sender will contain the most recent email source address to the MS. This value continually changes as email objects are received, therefore the window of opportunity for containing the value is quite unpredictable. Thus, having a condition solely on an AppTerm without regard for checking a WDR that triggers checking the configuration seems useless, however a MS may have many WDRs in process thereby reasonably causing frequent checks to M_sender. A more useful charter with an AppTerm will check the AppTerm against a WDR field or subfield, while keeping in mind that WDRs in process trigger testing the charter(s). For example:

(_appfld.email.source=M_sender)

or the equivalent of:

(M_sender=_appfld.email.source)

checks each WDR in process for containing an Application field **1100***k* from the email section (if available) which matches an AppTerm. While this again seems unusual since M_sender dynamically changes according to email objects received, timeliness of WDRs in process for MSs (e.g. in the wireless vicinity) can make this useful. Further, the programmer/user can specify more criteria for defining how close/far in the vicinity (e.g. atomic operators of $(range), (spec)$ (range), etc.

((_appfld.email.source=M_sender) & (_location $(500F)\loc_my))

The WDR in process is checked to see if the originating MS has a source email address that matches a most recently received email object and the MS is within 500 feet of my current location. This configuration can be useful, for example to automatically place a call to a friend when they just sent you an email and they are nearby. You can then walk over to them and converse about the email information. Good or poor configurations can be made. One embodiment of an internalizer warns a user when an awkward configuration has been made.

In looking at actions for this example, the command operand pair is for "Notify Indicator" with two parameters (M_sender, \thisms). M_sender is what to use for the indicator (the source address matched). Thus, an AppTerm can be used as a parameter. \thisms is an atomic term for this MS ID. If the expression evaluates to true, the MS hosting the charter configuration will be notified with an indicator text string (e.g. billj@iswtechnologies.com). Notify Indicator displays the indicator in the currently focused title bar text of a windows oriented interface. In another embodiment, Notify indicator command processing displays notification data in the focused user interface object at the time of being notified. The action has optional specifications for Description and History information to be generated (when internalized).

In general, History information will be updated as the user changes the associated configuration in the future, either in syntax (recognized on internalization (e.g. to data structures)), with FIGS. **38** through **48**B, etc.

```
(B_srchSubj ^ M_subject) & !(_fcnTest(B_srchSubj)) :
  "ms group"[G].Store DBobject(JOESDB.LBXTABS.TEST,
    "INSERT INTO TABLESAV (" && \thisMS && ", " && \timestamp
    &&
      ", 9);", \thisMS);
```

IF (the most recently specified B_srchSubj string is in (i.e. is a substring of) the most recently received email object M_subject (i.e. email subject string)), AND if (the invocation of the function _fcnTest( ) with the parameter of the most recently specified B_srchSubj string returns false) (i.e. ! the return code results in true), THEN the configured action after the colon (:) shall take place assuming there are applicable privileges configured as well. Again, keep in mind that WDRs in process (e.g. to queue **22**, outbound and/or inbound) provide the triggers upon which charters are tested, therefore the fact that no WDR field is specified in the conditions is strange, but make a good point. The example demonstrates using otherwise unrelated AppTerms and an invoked function (e.g. can be dynamically linked as in a Dynamic Link Library (DLL) or linked through an extern label _fcnTest). B_srchSubj contains the most recently specified search criteria string requested to the MS browser application. WDRTerm(s), AppTerm(s) and atomic terms can be used in conditions, as parameters, or as portions in any part of a configured charter.

The action demonstrates an interesting format for representing the optional Host construct (qualifier) of the BNF grammar for where the action should take place (assuming privilege to execute there is configured). "ms group"[G]. tells the internalizer to search for a group definition like an array and find the first member of the group meeting the subscript definition. This would be "George" (the G). Any substring of "George" (or the entire string) could have been used to indicate use George from the "ms group". This allows a shorthand reference to the item(s) of the group. Multiple members that match "G" would all apply for the action. Also, note that the double quotes are used whenever variables contain significant blanks. "ms group"[G].Store DBobject tells the internalizer that the Command Operand pair is to be executed at the George MS for storing to a database object per parameters. An equivalent form is George.Store DB-object with the Host specification explicitly specified as George. The parameters of (JOESDB.LBXTABS.TEST, "INSERT INTO TABLESAV ("'&& \thisMS &&'", "'&& \timestamp &&'", 9);", \thisMS) indicates to insert a row into the table TABLESAV of the TEST database at the system "this MS" (the MS hosting the configuration). The second (query) parameter matches the number of columns in the table for performing a database row insert. Like other compilers/interpreters, the " " evaluates to a single double quote character when double quotes are needed inside strings. A single quote can also be legal to delimit query string parameters (as shown). This example shows using atomic term(s) for a parameter (i.e. elaborates to underlying value; WDRTerm(s) can also be used for parameters). This example introduces a concatenation operator (&&) for concatenating together multiple values into a result string for one parameter (e.g. "INSERT INTO TABLESAV ('Bill', '20080421024421.45', 9);"). Other embodiments will sup-

port other programmatic operators in expressions for parameters. Still other embodiments will support any reasonable programmatic statements, operators, and syntax among charter configuration to facilitate a rich method for defining charters **12**.

Note that while we are configuring for the MS George to execute the action, we are still performing the insert to the MS hosting the Charter configuration (i.e. target system is \thisms). We could just as easily have configured:

```
Store DBobject(JOESDB.LBXTABS.TEST,
  "INSERT INTO TABLESAV (" && \thisMS && ", " && \timestamp &&
    ", 9);");
```

without using George to execute the action, and to default to the local MS. Privileges will have to be in place for running the action at the George MS with the original charter of FIG. **51**B.

```
(_I_msid="Sophia" & \loc_my (30M)$$(25M) _I_location):
  "ms group".Invoke App (alert.cmd);
```

_I_msid is a WDRTerm indicating to check the condition of the WDRs inbound to the local MS (e.g. deposited to receive queue **26**). The condition _I_msid="Sophia" tests if the inbound WDR has a WDR MS ID field **1100***a* equal to the MS ID Sophia. "=" is an atomic operator. & is a CondOp. _I_location is the contents of the inbound WDR location field **1100***c*, so that the condition of (\loc_my 30M$$25M _I_location) tests my current location (i.e. receiving MS) for being within 25 meters, within the last 30 minutes, of the location of the WDR received. A group is specified for where to run the action (i.e. Host specification), yet no member is referenced. The alert.cmd file is executed at each MS of the group (all three), provided there is a privilege allowing this MS to run this action there, and provided the alert.cmd file is found for execution (e.g. preferably uses PATH environment variable or similar mechanism; fully qualified path can specify).

```
(%c:\myprofs\interests.chk > 90):
  Send Email ("Howdy " && _I_msid && " !!\n\nOur profiles matched
  > 90%.\n\n"
    && "Call me at " && \appfld.phone.id && ". We are " &&
    (_I_location - \loc_my)F && " feet apart\n", \appfld.source.id,
    "Call Me!",
    ,, _I_appfld.email.source);
```

This example uses an atomic profile match operator (%). A profile is optionally communicated in Application field **1100***k* subfield _appfid.profile.contents. A user specifies which file represents his current profile and it is sent outbound with WDRs (see FIG. **78** for profile example). Upon receipt by a receiving MS, the current profile can be compared to the profile information in the WDR. (% c:\myprofs\interests.chk>90) provides a condition for becoming true when the hosting MS profile interests.chk is greater than 90% a match when matching to a WDR profile of field **1100***k* (preferably matches on a tag basis). The profile operator here is triggered on in process WDRs. An alternate embodiment will specify where to check the WDR (e.g. _I_%, _O_% or _%). If the expression evaluates to true, the Send Email (Command Operand pair) action is invoked with appropriate parameters. Note that the newline (\n) character and concatenation operator is used. Also, note the WDRTerm (_I_location) and atomic term (\loc_my) were used in an arithmetic statement to figure out the number of feet in distance between the location of the inbound WDR and "my current location". The result is automatically typecast to a string for the concatenation like most PPLs. The recipient is the email source in Application fields **1100***k*. The default email attributes are specified (,,).

In sum, there are many embodiments derived from the BNF grammar of FIG. **30**A through **30**E. FIGS. **51**A and **51**B are simple examples with some interesting syntactical feature considerations. Some embodiments will support programmatic statements intermingled with the BNF grammar syntax derivative used to support looping, arithmetic expressions, and other useful programmatic functionality integrated into Privilege and Charter definitions. FIGS. **51**A and **51**B illustrate a WPL for programming how a MS is to behave. WPL is a unique programming language wherein peer to peer interaction events containing whereabouts information (WDRs) provide the triggers for novel location based processing. Permissions and charters provide rules which govern the interoperable LBX processing between MSs. While WPL is more suited for a programmer type of user, the intent of this disclosure is to simplify configurations for all types of users. WPL may suit an advanced user while FIGS. **35**A through **37**C may suit more prevalent and novice users. Other embodiments may further simplify configurations. Some WPL embodiments will implement more atomic operators, AppTerm(s), WDRTerm(s) and other configurable terms without departing from the spirit and scope of this disclosure. It is the intent that less time be spent on documentation and more time be spent implementing it. Permissions and charters are preferably centralized to the MS, and maintained with their own user interface, outside of any particular MS application for supervisory control of all MS LBX applications. See FIG. **1**A for how PIP data **8** is maintained outside of other MS processing data and resources for centralized governing of MS operations.

In alternate embodiments, an action can return a return code/value, for example to convey success, failure, or some other value(s) back to the point of performing the action. A syntactical embodiment:

```
((_I_msid = "Brian") & (_I_location @ \loc_my) [D="multi-cond text";H;]):
  Notify Autodial (214-405-5422,,,, Invoke App (myscript.cmd ("B"))
    [T=20080302;]);
```

Based on an outcome from Invoke App (myscript . . .), the returned value is passed back and used as a parameter to Notify AutoDial. The Notify AutoDial executable spawned can then use the value at run-time to affect Notify processing. Invoke App may return a plurality of different values depending on the time the action is processed, and what the results are of that processing. Some parameters are specified to use defaults (i.e. ,,,,).

FIG. **52** depicts another preferred embodiment C programming source code header file contents, derived from the grammar of FIGS. **30**A through **30**E. FIG. **52** is more efficient for an internalized BNF grammar form by removing unnecessary data. When comparing FIG. **52** with FIGS. **34**E through **34**G, FIG. **52** has removed description and history information since this is not necessary for internalization/processing. A TIMESPEC is the same as defined at the top of FIG. **34**E, but time specification information has been merged to where it is needed, rather than keeping it in multiple places as configured for deducing a merged result later. There are many reasonable embodiments of a derivative of the BNF grammar of FIGS. **30**A through **30**E.

FIG. **53** depicts a preferred embodiment of a Prefix Registry Record (PRR) for discussing operations of the present disclosure. A PRR **5300** is for configuring which prefix is assigned to which application used in an AppTerm. This helps to ensure that an AppTerm be properly usable when referenced in a charter. A prefix field **5300***a* provides the prefix in an AppTerm syntax (e.g. M_sender such that "M" is the prefix). Any string can be used for a prefix (i.e. configured in field **5300***a*), but preferably there are a minimal number of characters to save syntax encoding space. A description field **5300***b* provides an optional user specified description for a PRR **5300**, but it may include defaulted data available with an application supporting at least one AppTerm. A service references field **5300***c* identifies, if any, the data processing system services associated with the application for the AppTerm referenced with the prefix of field **5300***a*. Validation of such services may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***c* potentially contains a list of service references. An application references field **5300***d* identifies, if any, data processing system application references (e.g. names) associated with the Application for the AppTerm referenced with the prefix of field **5300***a*. Validation of such applications referenced may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***d* potentially contains a list. A process references field **5300***e* identifies, if any, data processing operating system processes for spawning associated with the Application for the AppTerm referenced with the prefix of field **5300***a*. Validation of such processes may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***e* potentially contains a list. A paths field **5300***f* identifies, if any, data processing system file name paths to executables (e.g. .exe, .dll, etc) for spawning associated with the Application for the AppTerm referenced with the prefix of field **5300***a*. Validation of such paths may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***f* potentially contains a list. A documentary field **5300***g* documents each Application data variable (i.e. AppTerm data name without prefix), and an optional description, for what data is exposed for the Application which can be used in the AppTerm. Validation of data in field **5300***g* data may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***g* potentially contains a list. Extension field **5300***h* contains other data for how to test for whether or not the Application of the PRR is up and running in the MS, additional information for starting the Application, and additional information for accessing application vitals. Validation of information may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***h* may be a list, or null.

In one preferred embodiment, PRRs are supplied with a MS prior to user first MS use, and no administrator or user has to maintain them. In another embodiment, only a special administrator can maintain PRRs, which may or may not have been configured in advance. In another embodiment, a MS user can maintain PRRs, which may or may not have been configured in advance.

FIG. **54** depicts an example of an XML syntactical encoding embodiment of permissions and charters, derived from the BNF grammar of FIGS. **30**A through **30**E, for example as user specified, system maintained, system communicated, system generated, etc. Enough information is provided for those skilled in the art to define an appropriate XML syntax of the disclosed BNF grammar in light of disclosure heretofore. A simple embodiment of variables can be handled with a familiar Active Service Page (ASP) syntax wherein variables are defined prior to being instantiated with a special syntax (e.g. <%=varName %>). Double quotes can be represented within double quote delimited character strings by the usual providing of two double quotes for each double quote character position. Those skilled in the art of XML recognize there are many embodiments for XML tags, how to support subtags, and tag attributes within a tag's scope. FIG. **54** provides a simple reference using a real example. FIG. **54** illustrates a WPL for less advanced users.

The syntax "_location $(300M) \loc_my" is a condition for the WDR in process being within 300 Meters of the vicinity of my current location. Other syntax is identifiable based on previous discussions.

FIG. **55**A depicts a flowchart for describing a preferred embodiment of MS user interface processing for Prefix Registry Record (PRR) configuration. Block **5502** may begin as the result of an authenticated administrator user interface, authenticated user interface, or as initiated by a user. Block **5502** starts processing and continues to block **5504** where initialization is performed before continuing to block **5506**. Initialization may include initializing for using an SQL database, or any other data form of PRRs. Processing continues to block **5506** where a list of current PRRs are presented to the user. The list is scrollable if necessary. A user preferably has the ability to perform a number of actions on a selected/specified PRR from the list presented at block **5506**. Thereafter, block **5508** waits for a user action in response to presenting PRRs. Block **5508** continues to block **5510** when a user action has been detected. If block **5510** determines the user selected to modify a PRR, then the user configures the specified PRR at block **5512** and processing continues back to block **5506**. Block **5512** interfaces with the user for PRR **5300** alterations until the user is satisfied with changes which may or may not have been made. Block **5512** preferably validates to the fullest extent possible the data of PRR **5300**. If block **5510** determines the user did not select to modify a PRR, then processing continues to block **5514**. If block **5514** determines the user selected a PRR for delete, then block **5516** deletes the specified PRR, and processing continues back to block **5506**. Depending on an embodiment, block **5516** may also properly terminate the application fully described by the PRR **5300**. If block **5514** determines the user did not select to delete a PRR, then processing continues to block **5518**. If block **5518** determines the user selected to add a PRR, then the user adds a validated PRR at block **5520** and processing continues back to block **5506**. Block **5520** preferably validates to the fullest extent possible the data of PRR **5300**. Depending on an embodiment, block **5520** may also properly start the application described by the PRR **5300**. If block **5518** determines the user did not select to add a PRR, then processing continues to block **5522**. If block **5522** determines the user selected to show additional detail of a PRR, then block **5524** displays specified PRR details including those details not already displayed at block **5506** in the list. Processing continues back to block **5506** when the user is complete browsing details. If block **5522** determines the user did not want to browse PRR details, then processing continues to block **5526**. If block **5526** determines the user selected to enable/disable (toggle) a specified PRR, then block **5528** uses PRR **5300** to determine if the associated application is currently enabled (e.g. running) or disabled (e.g. not running). Upon determination of the current state of the application for the specified PRR **5300**, block **5528** uses the PRR **5300** to enable (e.g. start if currently not running)), or disable (e.g. terminate if currently running), the application described fully by the specified PRR, before continuing back to block **5506**. Block **5528** should ensure the

Application has been properly started, or terminated, before continuing back to block **5506**. If block **5526** determines the user did not want to toggle (enable/disable) a PRR described application, then processing continues to block **5530**. If block **5530** determines the user selected to display candidate AppTerm supported applications of the MS, then block **5532** presents a list of MS applications potentially configurable in PRR form. Block **5532** will interface with the user until complete browsing the list. One embodiment of block **5532** accesses current PRRs **5300** and displays the applications described. Another embodiment accesses an authoritative source of candidate AppTerm supported applications, any of which can be configured as a PRR. Processing continues back to block **5506** when the user's browse is complete. If block **5530** determines the user did not select to display AppTerm supported applications, then processing continues to block **5534**. If block **5534** determines the user selected to use a data source as a template for automatically populating PRRs **5300**, then block **5536** validates a user specified template, uses the template to alter PRRs **5300**, and processing continues back to block **5506**. PRRs may be optionally altered at block **5536** for replacement, overwrite, adding to, or any other alternation method in accordance with a user or system preference. In some embodiments, existing PRRs can be used for template(s). If block **5534** determines the user did not select to use a data source for a PRR template, then processing continues to block **5538**. If block **5538** determines the user did not select to exit PRR configuration processing, then block **5540** handles all other user actions detected at block **5508**, and processing continues back to block **5506**. If block **5538** determines the user did select to exit, then processing continues to block **5542** where configuration processing cleanup is performed before terminating FIG. **55**A processing at block **5544**. Depending on an embodiment, block **5542** may properly terminate data access initialized at block **5504**, and internalize PRRs for a well performing read-only form accessed by FIG. **55**B. Appropriate semaphore interfaces are used.

FIG. **55**A is used to expose those AppTerm variables which are of interest. Candidate applications are understood to maintain data accessible to charter processing. Different embodiments will utilize global variables (e.g. linked extern), dynamically linked variables, shared memory variables, or any other data areas accessible to both the application and charter processing with proper thread safe synchronized access.

FIG. **55**B depicts a flowchart for describing a preferred embodiment of Application Term (AppTerm) data modification. An application thread performing at least one AppTerm update uses processing of FIG. **55**B. A participating application thread starts processing at block **5552** as the result of a standardized interface, integrated processing, or some other appropriate processing means. Block **5552** continues to block **5554** where an appropriate semaphore lock is obtained to ensure synchronous data access between the application and any other processing threads (e.g. charter processing). Processing then continues to block **5556** for accessing the application's associated PRR (if one exists). Thereafter, if block **5558** determines the PRR exists and at least one of the data item(s) for modification are described by field **5300***g*, block **5560** updates the applicable data item(s) described by field **5300***g* appropriately as requested by the application invoking FIG. **55**B processing. Thereafter, block **5562** releases the semaphore resource locked at block **5554** and processing terminates at block **5564**.

If block **5558** determines the associated PRR was not found or all data items of the found PRR for modification are not described by field **5300***g*, then processing continues directly to block **5562** for releasing the semaphore lock, thereby performing no updates to an AppTerm. PRRs **5300** control eligibility for modification by applications, as well as which AppTerm references can be made in charter processing.

An AppTerm is accessed (read) by grammar processing with the same semaphore lock control used in FIG. **55**B.

FIG. **56** depicts a flowchart for appropriately processing an encoding embodiment of the BNF grammar of FIGS. **30**A through **30**E, in context for a variety of parser processing embodiments. Those skilled in the art may take information disclosed heretofore to generate table records of FIGS. **35**A through **37**C, and/or data of FIGS. **34**A through **34**G (and/or FIG. **52**), and/or datastreams of FIG. **33**A through **33**C, and/or a suitable syntax or internalized form derivative of FIGS. **30**A through **30**E. Compiler, interpreter, data receive, or other data handling processing as disclosed in FIG. **56** is well known in the art. Text books such as "Algorithms+Data Structures=Programs" by Nicklaus Wirth are one of many for guiding compiler/interpreter development. A BNF grammar of FIGS. **30**A through **30**E may also be "plugged in" to a Lex and Yacc environment to isolate processing from parsing in an optimal manner. Compiler and interpreter development techniques are well known. FIG. **56** can be viewed in context for adapting Permission and Charter processing to an existing source code processing environment (e.g. within PPLs). FIG. **56** can be viewed in context for new compiler and interpreter processing of permissions and/or charters (e.g. WPL). FIG. **56** can be viewed in context for receiving Permission and/or Charter data (e.g. syntax, datastream, or other format) from some source (e.g. communicated to MS). FIG. **56** can be viewed in context for plugging in isolated Permission and Charter processing to any processing point of handling a derivative encoding of the BNF grammar of FIGS. **30**A through **30**E.

Data handling of a source code for compiling/interpreting, an encoding from a communication connection, or an encoding from some processing source starts at block **5602**. At some point in BNF grammar derived data handling, a block **5632** gets the next (or first) token from the source encoding. Tokens may be reserved keywords, delimiters, variable names, expression syntax, or some construct or atomic element of an encoding. Thereafter, if block **5634** determines the token is a reserved key or keyword, block **5636** checks if the reserved key or keyword is for identifying permissions **10** (e.g. FIG. **51**A "Permissions", FIG. **54** "permission", FIG. **33**B Permissions/Permission, etc), in which case block **5638** sets a stringvar pointer to the entire datastream representative of the permission(s) **10** to be processed, and block **5640** prepares parameters for invoking LBX data internalization processing at block **5642**.

If block **5636** determines the reserved key or keyword is not for permission(s) **10**, then processing continues to block **5646**. Block **5646** checks if the reserved key or keyword is for identifying charters **12** (e.g. FIG. **51**B "Charters", FIG. **54** "charter", FIG. **33**C Charters/Charter, etc), in which case block **5648** sets a stringVar pointer to the entire datastream representative of the charter(s) **12** to be processed, and block **5650** prepares parameters for invoking LBX data internalization processing at block **5642**.

Blocks **5640** and **5650** preferably have a stringVar set to the permission/charter data encoding start position, and then set a length of the permission/charter data for processing by block **5642**. Alternatively, the stringVar is a null terminated string for processing the permission(s)/charter(s) data encoding. Embodiment requirements are for providing appropriate parameters for invoking block **5642** for unambiguous processing of the entire permission(s)/charter(s) for parsing and

processing. The procedure of block **5642** has already been described throughout this disclosure (e.g. creating a processable internalized form (e.g. database records, programmatic structure, etc)). Upon return from block **5642** processing, block **5644** resets the parsing position of the data source encoding provided at block **5602** for having already processed the permission(s)/charter(s) encoding handled by block **5642**. Thereafter, processing continues back to block **5632** for getting the next token from the data encoding source.

If block **5646** determines the reserved key or keyword is not for charter(s) **12**, then processing continues to process the applicable reserved key or keyword identified in the source data encoding. If block **5634** determines the token is not a reserved key or keyword, then processing continues to the appropriate block for handling the token which is not a reserved key or keyword. In any case there may be processing of other source data encoding not specifically for a permission or charter.

Eventually, processing continues to a block **5692** for checking if there is more data source to handle/process. If block **5692** determines there is more data encoding source, processing continues back to block **5632** for getting the next token. If block **5692** determines there is no more data encoding source, processing continues to block **5694** for data encoding source processing completion, and then to block **5696** for termination of FIG. **56** processing.

Depending on the embodiment, block **5694** may complete processing for:

- Compiling one of the PPLs (or other programming language) with embedded/integrated encodings for permissions **10** and/or charters **12**;
- Interpreting one of the PPLs (or other programming language) with embedded/integrated encodings for permissions **10** and/or charters **12**;
- Receiving the encoding source data from a communications channel;
- Receiving the encoding source data from a processing source;
- Receiving the encoding source data from a user configured source;
- Receiving the encoding source data from a system configured source; or
- Internalizing, compiling, interpreting, or processing an encoding derived from the disclosed BNF grammar for Permissions **10** and/or Charter **12**.

Blocks **5636** through **5650** may represent plug-in processing for permissions **10** and/or charters **12**. Depending on when and where processing occurs for FIG. **56**, appropriate semaphores may be used to ensure data integrity.

LBX: Permissions and Charters—WDR Processing

As WDR information is transmitted/received between MSs, privileges and charters are used to govern automated actions. Thus, privileges and charter govern processing of at least future whereabouts information to be processed. There is WDR In-process Triggering Smarts (WITS) in appropriate executable code processing paths. WITS provides the intelligence of whether or not privilege(s) and/or charter(s) trigger(s) an action. WITS is the processing at a place where a WDR is automatically examined against configured privileges and charter to see what actions should automatically take place. There are three different types of WITS, namely: maintained WITS (mWITS), inbound WITS (iWITS), and outbound WITS (oWITS). Each type of WITS is placed in a strategic processing path so as to recognize the event for when to process the WDR. Maintained WITS (mWITS) occur at those processing paths applicable to a WDR in process for being maintained at an MS (e.g. inserted to queue **22**). Other embodiments may define other maintained varieties of a WDR in process for configurations (e.g. inbound, outbound, in-process2Q22, in-process2History (i.e. WDR in process of being maintained to LBX history **30**), in-process2application(s) (i.e. WDR in process of being maintained/communicated to an application), etc). Inbound WITS (iWITS) occur at those processing paths applicable to a WDR which is inbound to a MS (e.g. communicated to the MS). Outbound WITS (oWITS) occur at those processing paths applicable to a WDR which is outbound from a MS (e.g. sent by an MS). There are various WITS embodiments as described below. Users should keep in mind that a single WDR may be processed multiple times (by different WITS) with configuring charters that refer to different WITS (e.g. first inbound, then to queue **22**). One embodiment supports only mWITS. Another embodiment supports only iWITS. Another embodiment supports oWITS. Yet another embodiment supports use of any combination of available WITS.

mWITS:

The preferred embodiment is a new block **273** in FIG. **2**F such that block **272** continues to block **273** and block **273** continues to block **274**. This allows mWITS processing block **273** to see all WDRs which are candidate for insertion to queue **22**, regardless of the role check at block **274**, confidence check at block **276**, and any other FIG. **2**F processing. In some embodiments, block **273** may choose to use enabled roles and/or confidence and/or any WDR field(s) values and/or permissions and/or any other processing result to decisively affect whether or not the WDR should be examined and/or processed further by FIG. **2**. For example, block **273** may result in processing to continue directly to block **294** or **298** (rather than block **274**). For example, upon determining that the WDR source had not provided any privileges to the receiving MS, the WDR can be ignored so as to not use resources of the MS. In another example, a WDR shows that it arrived completely wirelessly (e.g. field(s) **1100***f*) and did not go through an intermediary service (e.g. router). The WDR may provide usefulness in locating the receiving MS despite the receiving MS not being privileged by the source MS, in which case block **273** continues to block **274** for WDR processing. It may be important to filter WDRs so that only those WDRs are maintained which either a) contribute to locating (per configurations), or b) are associated with active permissions or charters for applicable processing. The WRC discussed above may also be used to cause block **273** to continue to block **294** or **298**. Such filtering is referred to as WITS filtering. WITS filtering may be crucial in a LBX architecture which supports MSs great distances from each other since there can be an overloading number of WDRs to process at any point in time. Charters and privileges that are configured are used for deciding which WDRS are to be "seen" (processed) further by FIG. **2**F processing. If there are no privileges and no charters in effect for the in process WDR, then the WDR may be ignored. If there is no use for the WDR to help locate the receiving MS, then the WDR may also be ignored. If there are privileges and charters in effect for the in process WDR, then the WDR can be processed further by FIG. **2**F, even if not useful for locating the MS.

One preferred embodiment does make use of the confidence field **1100***d* to ensure the peer MS has been sufficiently located. Block **273** will compare information of the WDR with configured privileges to determine which

actions should be performed. When appropriate privileges are in place, block **273** will also compare information of the WDR with configured and privileged charters (e.g. _fldname) to determine applicable configured charter actions to be performed.

Alternate embodiments can move mWITS at multiple processing places subsequent to where a WDR is completed by the MS (e.g. blocks **236**, **258**, **334**, **366**, **418**, **534**, **618**, **648**, **750**, **828**, **874**, **958**, **2128**, **2688**, etc).

Another embodiment can support mWITS at processing places subsequent to processing by blocks **1718** and **1722** to reflect user maintenance.

Yet another embodiment recognizes in mWITS that the WDR was first inbound to the MS and is now in process of being maintained (e.g. to queue **22**). This can allow distinguishing between an inbound WDR, maintained WDR, and inbound AND maintained WDR. In one embodiment, the WDR (e.g. field **1100***g*) carries new bit(s) of information (e.g. set by receive processing when inserting to queue **26**) for indicating the WDR was inbound to the MS. The new bit(s) are checked by mWITS for new processing (i.e. inbound AND maintained WDR).

iWITS:

The preferred embodiment is a new block **2111** in FIG. **21** such that block **2110** continues to block **2111** (i.e. on No condition) and block **2111** continues to block **2112**. This allows iWITS processing block **2111** to see all inbound WDRs, regardless of the confidence check at block **2114**, and any other FIG. **21** processing. In some embodiments, block **2111** may choose to use confidence and/or any WDR field(s) and/or permissions and/or any other processing result to decisively affect whether or not the WDR should be examined and/or processed further by FIG. **21**. Block **2111** may result in processing to continue directly to block **2106** (rather than block **2112**). For example, upon determining that the WDR source had not provided any privileges to the receiving MS, the WDR can be ignored so as to not use resources of the MS. In another example, a WDR shows that it arrived completely wirelessly (e.g. field(s) **1100***f*) and did not go through an intermediary service (e.g. router). The WDR may provide usefulness in locating the receiving MS despite the receiving MS not being privileged by the source MS, in which case block **2111** continues to block **2112** for WDR processing. Similar WITS filtering can occur here as was described for mWITS processing above, with the advantage of intercepting WDRs of little value at the earliest possible time and preventing them from reaching subsequent LBX processing.

One preferred embodiment does make use of the confidence field **1100***d* to ensure the peer MS has been sufficiently located. Block **2111** will compare information of the WDR with configured privileges to determine which actions should be performed. When appropriate privileges are in place, block **2111** will also compare information of the WDR with configured and privileged charters (e.g. _I_fldname) to determine applicable configured charter actions to be performed.

Another embodiment can support iWITS at processing places associated with receive queue **26**, for example processing up to the insertion of the WDR to queue **26**.

oWITS:

The preferred embodiment incorporates a new block **2015** in FIG. **20** such that block **2014** continues to block **2015** and block **2015** continues to block **2016**. This allows oWITS processing block **2015** to see all its outbound WDRs for FIG. **20** processing. In some embodiments, block **2015** may choose to use confidence and/or any WDR field(s) and/or permissions and/or any other processing result to decisively affect whether or not the WDR should be processed further by FIG. **20**. Block **2015** may result in processing to continue directly to block **2018**. The WRC discussed may also be used appropriately here. Similar WITS filtering can occur here as was described for mWITS and iWITS processing above, with the advantage of intercepting WDRs of little value to anyone else in the LN-expanse, and preventing the WDRs from reaching subsequent LBX processing at remote MSs that will have no use for them.

The preferred embodiment will also incorporate a new block **2515** in FIG. **25** such that block **2514** continues to block **2515** and block **2515** continues to block **2516**. This allows oWITS processing block **2515** to see all its outbound WDRs of FIG. **25** processing. In some embodiments, block **2515** may choose to use confidence and/or any WDR field(s) and/or permissions and/or any other processing result to decisively affect whether or not the WDR should be examined and/or processed further by FIG. **25**. Block **2515** may result in processing to continue directly to block **2506**. For example, upon determining that the WDR is destined for a MS with no privileges in place, the WDR can be ignored and unprocessed (i.e. not sent). The WRC discussed may also be used appropriately here. Similar WITS filtering can occur here as was described for mWITS, iWITS and oWITS processing above, with the advantage of intercepting WDRs of little value to anyone else in the LN-expanse, and preventing the WDRs from reaching subsequent LBX processing at remote MSs that will have no use for them.

Blocks **2015** and **2515** will compare information of the WDR with configured privileges to determine which actions should be performed. When appropriate privileges are in place, blocks **2015**/**2515** will also compare information of the WDR with configured charters (e.g. _O_fldname) to determine applicable configured and privileged charter actions to be performed.

Another embodiment can support oWITS at processing places associated with send queue **24**, for example after the insertion of the WDR to queue **24**.

Yet another embodiment recognizes in oWITS that the WDR was first maintained to the MS and is now in process of being sent outbound. This can allow distinguishing between an outbound WDR, maintained WDR, and outbound AND maintained WDR. Different embodiments will use different criteria for what designates an outbound AND maintained WDR, for example seeking certain values in maintained WDR field(s), seeking certain values in outbound WDR field(s), or both. In one embodiment, the WDR carries new bit(s) of information (e.g. set by send processing) for indicating the WDR was outbound from the MS. WDR processing for a maintained WDR and/or an outbound WDR can also be made relevant for designating an outbound AND maintained WDR. Criteria may be important in this embodiment since an outbound WDR was maintained in some fashion prior to being candidate as an outbound WDR.

FIG. **57** depicts a flowchart for describing a preferred embodiment of WDR In-process Triggering Smarts (WITS) processing. The term "Triggering Smarts" is used to describe intelligent processing of WDRs for privileges and/or charters that may trigger configured processing such as certain

actions. FIG. **57** is presented to cover the different WITS embodiments discussed above. WITS processing is of PIP code **6**, and starts at block **5700** with an in-process WDR as the result of the start of new blocks **273**, **2111**, **2015** and **2515** (as described above). While preferred WITS embodiments include new blocks **273**, **2111**, **2015**, and **2515**, it is to be understood that alternate embodiments may include FIG. **57** processing at other processing places, for example as described above. There are similarities between mWITS, iWITS and oWITS. FIG. **57** is presented in context for each WITS type. Thus, block **5700** shall be presented as being invoked for mWITS, iWITS, and oWITS in order to process a WDR (i.e. in-process WDR) that is being maintained to the MS of FIG. **57** processing (e.g. to queue **22**), is inbound to the MS of FIG. **57** processing, and/or is outbound from the MS of FIG. **57** processing. Applicable charter configurations (_ref, _I_ref, _O_ref) and applicable privileges are to be handled accordingly.

Block **5700** continues to block **5702**-*a* where the WRC and applicable origination information of the WDR is accessed. Thereafter, if the WRC and WDR information indicates to ignore the WDR at block **5702**-*b*, then processing continues to block **5746**, otherwise processing continues to block **5704**. Whenever block **5746** is encountered, the decision is made (assumed in FIG. **57**) to continue processing the WDR or not continue processing the WDR in processing which includes FIG. **57** (i.e. FIGS. **2**F, **20**, **21** **25**) as described above. This decision depends on how block **5746** was arrived to by FIG. **57** processing.

Block **5704** determines the identity (e.g. originating MS) of the in-process WDR (e.g. check field **1100***a*). Thereafter, if block **5706** determines the identity of the in-process WDR does not match the identity of the MS of FIG. **57** processing, processing continues to block **5708**. Block **5706** continues to block **5708** when a) the in-process WDR is from other MSs and is being maintained at the MS of FIG. **57** processing (i.e. FIG. **57**=mWITS); or b) the in-process WDR is from other MSs and is inbound to the MS of FIG. **57** processing (i.e. FIG. **57**=iWITS). For example, a first MS of FIG. **57** processing handles a WDR from a second MS starting at block **5708**.

With reference now to FIG. **58**, depicted is an illustration for granted data characteristics in the present disclosure LBX architecture, specifically with respect to granted permission data and granted charter data as maintained by a particular MS of FIG. **57** processing (i.e. as maintained by "this MS"). To facilitate discussion of FIG. **57**, permission data **10** can be viewed as permission data collection **5802** wherein arrows shown are to be interpreted as "provides privileges to" (i.e. Left Hand Side (LHS) provides privileges to the Right Hand Side (RHS)). Any of the permissions representations heretofore described (internalized, datastream, XML, source code, or any other BNF grammar derivative) can be used to represent, or encode, data of the collection **5802**. Regardless of the BNF grammar derivative/representation deployed, the minimal requirement of collection **5802** is to define the relationships of privileges granted from one ID to another ID (and perhaps with associated MSRelevance and/or TimeSpec qualifier(s)). Whether grants or explicit privileges are assigned, ultimately there are privileges granted from a grantor ID to a grantee ID.

Different identity embodiments are supported (e.g. MS ID or user ID) for the LHS and/or RHS (see BNF grammar for different embodiments). Permission data collection **5802** is to be from the perspective of one particular MS, namely the MS of FIG. **57** processing. Thus, the terminology "this MS ID" refers to the MS ID of the MS of FIG. **57** processing. The terminology "WDR MS ID" is the MS ID (field **1100***a*) of an in-process WDR of FIG. **57** processing distinguished from all other MS IDs configured in collection **5802** at the time of processing the WDR. The terminology "other MS IDs" is used to distinguish all other MS IDs configured in collection **5802** which are not the same as the MS ID of the terminology "WDR MS ID" (i.e. MS IDs other than the MS ID (field **1100***a*) of the in-process WDR of FIG. **57** processing (also other than the "this MS" MS ID)). Privilege configurations **5810** are privileges provided from an in-process WDR MS ID (i.e. WDR being processed by FIG. **57** at "this MS") to the MS ID of FIG. **57** processing. The groups an ID belongs to can also provide, or be provided with, privileges so that the universe of privileges granted should consider groups as well. Privilege configurations **5820** are privileges provided from the MS of FIG. **57** processing (this MS) to the MS ID (field **1100***a*) of the in-process WDR being processed by FIG. **57**. Privilege configurations **5830** are privileges provided from the MS of FIG. **57** processing (this MS) to MS IDs (field **1100***a*) configured in collection **5802** other than the MS ID of the in-process WDR being processed by FIG. **57** (also other than the "this MS" MS ID). Privilege configurations **5840** are privileges provided from MS IDs configured in collection **5802** at the MS of FIG. **57** processing (this MS) which are different than the MS ID of the in-process WDR being processed by FIG. **57** (also different than the "this MS" MS ID).

Also to facilitate discussion of FIG. **57**, charter data **12** can be viewed as a charter data collection **5852** wherein arrows shown are to be interpreted as "creates enabled charters for" (i.e. Left Hand Side (LHS) creates enabled charters for the Right Hand Side (RHS)). Any of the charter representations heretofore described (internalized, datastream, XML, source code, or any other BNF grammar derivative) can be used to represent, or encode, data of the collection **5852**. Regardless of the BNF grammar derivative/representation deployed, the minimal requirement of collection **5852** is to define the charters granted by one ID to another (and perhaps with associated TimeSpec qualifier(s); TimeSpec may be an aggregate-result of TimeSpec specified for the charter, charter expression, charter condition and/or charter term). Preferably, for charters with multiple actions, each action is evaluated on its own specified TimeSpec merit if applicable. In embodiments that use a tense qualifier in TimeSpecs: LBX history, appropriate queue(s), and any other reasonable source of information shall be utilized appropriately.

Different identity embodiments are supported (e.g. MS ID or user ID) for the LHS and/or RHS (see BNF grammar for different embodiments). A privilege preferably grants the ability to create effective (enabled) charters for one ID from another ID. However, in some embodiments the granting of a charter by itself from one ID to another ID can be treated like the granting of a permission/privilege to use the charter, thereby preventing special charter activating permission(s) be put in place. Charter data collection **5852** is also to be from the perspective of the MS of FIG. **57** processing. Thus, the terminology "this MS ID" refers to the MS ID of the MS of FIG. **57** processing. The terminology "WDR MS ID" is the MS ID (field **1100***a*) of the in-process WDR of FIG. **57** processing distinguished from all other MS IDs configured in collection **5852** at the time of processing the WDR. The terminology "other MS IDs" is used to distinguish all other MS IDs configured in collection **5852** which are not the same as the MS ID of the terminology "WDR MS ID" (i.e. MS IDs other than the MS ID (field **1100***a*) of the in-process WDR of FIG. **57** processing (also other than the "this MS" MS ID)). Charter configurations **5860** are charters created by the MS ID of an in-process WDR (i.e. WDR being processed by FIG. **57** at "this MS") for being effective at the MS of FIG. **57** processing

(this MS ID). The groups an ID belongs to can also provide, or be provided with, charters so that the universe of charters granted should consider groups as well. Charter configurations **5870** are charters created by the MS ID of FIG. **57** processing (i.e. this MS) for being effective at the MS of FIG. **57** processing (this MS ID). Charter configurations **5870** include the most common embodiments of creating charters for yourself at your own MS. Charter configurations **5880** are charters created by the MS ID of FIG. **57** processing (this MS) for being effective at MSs with MS IDs configured in collection **5852** other than the MS ID of the in-process WDR being processed by FIG. **57**. Charter configurations **5890** are charters at the MS of FIG. **57** processing (this MS) which are created by MS IDs other than the MS ID of the in-process WDR being processed by FIG. **57** (also other than the "this MS" MS ID).

Any subset of data collections **5802** and **5852** can be resident at a MS of FIG. **57** processing, depending on a particular embodiment of the present disclosure, however preferred and most common data used is presented in FIG. **57**. While FIG. **58** facilitates flowchart descriptions and discussions for in-process WDR embodiments of being maintained (e.g. to queue **22**), being inbound (e.g. communicated to the MS), and/or being outbound (e.g. communicated from the MS), FIGS. **49**A and **49**B provide relevant discussions for WDR in-process embodiments when considering generally "incoming" WDRs (i.e. being maintained (e.g. to queue **22**) or being inbound (e.g. communicated to the MS)).

In the preferred embodiment, groups defined local to the MS are used for validating which data using group IDs of collections **5802** and **5852** are relevant for processing. In alternate embodiments, group information of other MSs may be "visible" to FIG. **57** processing for broader group configuration consideration, either by remote communications, local maintaining of MS groups which are privileged to have their groups maintained there (communicated and maintained like charters), or another reasonable method.

With reference back to FIG. **57**, block **5708** forms a PRIVS2ME list of configurations **5810** and continues to block **5710** for eliminating duplicates that may be found. Block **5708** may collapse grant hierarchies to form the list. Duplicates may occur for privileges which include the duplicated privileges (i.e. subordinate privileges). For example, \lbxall specifies all LBX privileges and \nearar is only one LBX privilege already included in \lbxall. Recall that some privileges can be higher order scoped (subordinate) privileges for a plurality of more granulated privileges. Block **5710** additionally eliminates duplicates that may exist for permission embodiments wherein a privilege can enable or disable a feature. In a present disclosure embodiment wherein a privilege can enable, and a privilege can disable the same feature or functionality, there is preferably a tie breaker of disabling the feature (i.e. disabling wins). In an alternate embodiment, enabling may break a tie of ambiguity. Block **5710** further eliminates privileges that have a MSRelevance qualifier indicating the MS of FIG. **57** processing is not supported for the particular privilege, and also eliminates privileges with a TimeSpec qualifier invalid for the time of FIG. **57** processing (an alternate embodiment can enforce TimeSpec interpretation at blocks **5734** (i.e. in FIG. **59** processing) and **5736** (i.e. in FIG. **60** processing)). Thereafter, block **5712** forms a PRIVS2WDR list of configurations **5820** and continues to block **5714** for eliminating duplicates that may be found in a manner analogous to block **5710** (i.e. subordinate privileges, enable/disable tie breaker, MSRelevance qualifier, TimeSpec qualifier). Block **5712** may collapse grant hierarchies to form the list. An alternate embodiment can enforce TimeSpec interpretation at block **5738** (i.e. in FIG. **60** processing). Thereafter, block **5716** forms a CHARTERS2ME list of configurations **5860** and preferably eliminates variables by instantiating/elaborating at points where they are referenced. Then, block **5718** eliminates those charters which are not privileged. In some embodiments, block **5718** is not necessary (**5716** continues to **5720**) because un-privileged charters will not be permitted to be present at the MS of FIG. **57** processing anyway (e.g. eliminated when receiving). Nevertheless, block **5718** removes from the CHARTERS2ME list all charters which do not have a privilege (e.g. using PRIVS2WDR) granted by the MS (the MS user) of FIG. **57** processing to the creator of the charter, for permitting the charter to be "in effect" (activated). In the preferred embodiment, there is a privilege (e.g. \chrtrs) which can be used to grant the permission of activating any charters of another MS (or MS user) at the MS of FIG. **57** processing. In the preferred embodiment, there can be any number of subordinate charter privileges (i.e. subordinate to \chrtrs) for specifically indicating which type of charters are permitted. For example, privileges for governing which charters are to be active from a remote MS include:

- mWITS specifications (allow charters with _fldname);
- iWITS specifications (allow charters with _I_fldname);
- oWITS specifications (allow charters with _O_fldname);
- specified atomic terms (e.g. a privilege for each eligible atomic term use);
- specified WDRTerms (e.g. a privilege for each eligible WDRTerm use);
- specified AppTerms (e.g. a privilege for each eligible AppTerm use);
- specified operators (e.g. a privilege for each eligible atomic operator use);
- specified conditions;
- specified actions;
- specified host targets for actions; and/or
- any identifiable characteristic of a charter encoding as defined in the BNF grammar of FIGS. **30**A through **30**E.

In any embodiment, block **5718** ensures no charters from other users are considered active unless appropriately privileged (e.g. using PRIVS2WDR). Thereafter, block **5720** forms a MYCHARTERS list of configurations **5870** and preferably eliminates variables by elaborating at points where they are referenced, before continuing to block **5732**.

Block **5732** checks the PRIVS2ME list to see if there is a privilege granted from the identity of the in-process WDR to the MS (or user of MS) of FIG. **57** processing for being able to "see" the WDR. One main privilege (e.g. \lbxiop) can enable or disable whether or not the MS of FIG. **57** processing should be able to do anything at all with the WDR from the remote MS. If block **5732** determines this MS can process the WDR, then processing continues to block **5734**. Block **5734** enables local features and functionality in accordance with privileges of the PRIVS2ME list by invoking the enable features and functionality procedure of FIG. **59** with the PRIVS2ME list, and the in-process WDR as parameters (preferably passed by pointer/reference).

With reference now to FIG. **59**, depicted is a flowchart for describing a preferred embodiment of a procedure for enabling LBX features and functionality in accordance with a certain type (category) of permissions. Blocks **5920**, **5924**, **5928**, **5932**, **5936**, **5940**, **5944**, and **5946** enable or disable LBX features and functionality for semantic privileges. Processing of block **5734** starts at block **5900** and continues to block **5902** where the permission type list parameter passed (i.e. PRIVS2ME (**5810**) when invoked from block **5734**) is determined, and the in-process WDR may be accessed. The

list parameter passed provides not only the appropriate list to FIG. **59** processing, but also which list configuration (**5810**, **5820**, **5830** or **5840**) has been passed for processing by FIG. **59**. There are potentially thousands of specific privileges that FIG. **59** can handle. Therefore, FIG. **59** processing is shown to generically handle different classes (categories) of privileges, namely privilege classes of: privilege-configuration, charter-configuration, data send, impersonation, WDR processing, situational location, monitoring, LBX, LBS, and any others as handled by block **5946**. Privileges disclosed throughout the present disclosure fall into one of these classes handled by FIG. **59**.

Block **5902** continues to block **5904** where if it is determined that a privilege-configuration privilege is present in the list parameter passed to FIG. **59** processing, then block **5906** will remove privileges from the list parameter if appropriate to do that. For example, a privilege (or absence thereof detected in the list parameter for indicating no privileges can be defined/enabled in context of the list parameter causes block **5906** to remove all privileges from the list parameter and also from permissions **10** (i.e. **5810** of collection **5802** when FIG. **59** invoked from block **5734**). Similarly, any more granular privilege-configuration privileges of the list parameter causes processing to continue to block **5906** for ensuring remaining privileges of the list parameter (and of permissions **10** configurations) are appropriate. There can be many different privilege-configuration privileges for what can, and can't, be defined in permissions **10**, for example by any characteristic(s) of permissions data **10** according to the present disclosure BNF grammar. Block **5906** continues to block **5908** when all privilege-configuration privileges are reflected in the list parameter and collection **5802** of permissions **10**. If block **5904** determines there are no privilege-configuration privileges to consider in the list parameter passed to FIG. **59** processing, then processing continues to block **5908**.

Block **5908** gets the next individual privilege entry (or the first entry upon first encounter of block **5908** for an invocation of FIG. **59**) from the list parameter and continues to block **5910**. Blocks **5908** through **5946** iterate all individual privileges (list entries) associated with the list parameter of permissions **10** provided to block **5908**. If block **5910** determines there was an unprocessed privilege entry remaining in the list parameter (i.e. **5810** of collection **5802** when FIG. **59** invoked from block **5734**), then the entry gets processed starting with block **5912**. If block **5912** determines the entry is a charter-configuration privilege, then block **5914** will remove charters from CHARTERS2ME if appropriate to do that. For example, a privilege (or absence thereof detected in the list parameter for indicating no CHARTERS2ME charters can be defined/enabled in context of the list parameter causes block **5914** to remove all charters from CHARTERS2ME and also from charters **12** (i.e. **5860** of collection **5852** when FIG. **59** invoked from block **5734**). Similarly, any more granular charter-configuration privileges of the list parameter causes processing to continue to block **5914** for ensuring remaining charters of CHARTERS2ME (and of charters **12** configurations) are appropriate. There can be many different charters-configuration privileges for what can and can't be defined in charters **12**, for example by any characteristic(s) of charters data **12** according to the present disclosure BNF grammar, in particular for an in-process WDR from another MS. Any aspect of charters can be privileged (all, certain commands, certain operands, certain parameters, certain values of any of those, whether can specify Host for action processing, certain conditions and/or terms—See BNF grammar). Block **5914** then continues to block **5916**. Block **5916** will remove charters from MYCHARTERS if appropriate to do that. For example, a privilege (or absence thereof) detected in the list parameter for indicating certain MYCHARTERS charters (e.g. those that involve the in-process WDR) can/cannot be defined/enabled in context of the list parameter causes block **5916** to remove charters from MYCHARTERS for subsequent FIG. **57** processing. Changes to charters **12** for the MYCHARTERS list does not occur. This prevents deleting charters locally at the MS that the user spent time creating at his MS. Removing from the MYCHARTERS list is enough to affect subsequent FIG. **57** processing, for example of an in-process WDR. Block **5914** shown does additionally remove from charters **12** because the charters are not valid from a remote user anyway. One preferred embodiment to block **5914** will not alter charters **12** (only CHARTERS2ME) similarly to block **5916** so that subsequent FIG. **57** processing continues properly while preventing a remote MS user from resending charters (use of FIGS. **44**A and **44**B) at a subsequent time for reinstatement upon discovering the "this MS" FIG. **57** processing user had not provided a needed permission/privilege. Block **5916** continues back to block **5908** for the next entry. Blocks **5914** and **5916** make use of the privilege entry data from block **5908** (e.g. grantor ID, grantee ID, privilege, etc) to properly affect change of CHARTERS2ME and MYCHARTERS. CHARTERS2ME and MYCHARTERS are shown as global variables accessible from FIG. **57** processing to FIG. **59** processing, but an alternate embodiment will pass these lists as additional parameters determined at block **5902**. If block **5912** determined the currently iterated privilege is not a charter configuration privilege, then processing continues to block **5918**.

If block **5918** determines the entry is a data send privilege, then block **5920** will enable LBX features and functionality appropriately in context for the list parameter, and processing continues back to block **5908**. A data send privilege may be one that is used at block **4466** and enforced at block **4470** for exactly what data can or cannot be received. Any granulation of permission data **10** or charter data **12** (e.g. by any characteristic(s)) may be supported. A data send privilege may overlap with a privilege-configuration privilege or a charter-configuration privilege since either may be used at blocks **4466** and **4470**, depending on an embodiment. It may be useful to control what data can be received by a MS at blocks **4466** and **4470** versus what data actually gets used for FIG. **57** processing as controlled by blocks **5904**, **5906**, **5912**, **5914**, and **5916**. If block **5918** determines the entry is not a data send privilege, then processing continues to block **5922**. Data send privileges can control what privilege, charter, and/or group data can and cannot be sent to a MS (i.e. received by a MS). Data send privileges can be overall privileges, subordinate privileges, and/or privileges for any granulation of data based on type, size, value, age, or any other characteristic(s) available from a derivative of the BNF grammar of FIGS. **30**A through **30**E.

If block **5922** determines the entry is an impersonation privilege, then block **5924** will enable LBX features and functionality appropriately in context for the list parameter, and processing continues back to block **5908**. An impersonation privilege is one that is used to access certain authenticated user interfaces, some of which were described above. Any granulation of permission data **10** (e.g. by any characteristic(s)) may be supported, for example for any subset of MS user interfaces with respect to the present disclosure. Block **5924** may access security, or certain application interfaces accessible to the MS of FIG. **59** processing for read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to manage associated identity impersonation at the MS. If block **5922** determines the entry is not an impersonation

privilege, then processing continues to block **5926**. Impersonation privileges can be overall privileges, subordinate privileges, and/or privileges for any granulation of identity data or any other characteristic(s) available from a derivative of the BNF grammar of FIGS. **30**A through **30**E.

If block **5926** determines the entry is a WDR privilege, then block **5928** will enable LBX features and functionality appropriately in context for the list parameter, and processing continues back to block **5908**. A WDR privilege is one that is used to govern access to certain fields of the in-process WDR. Any granulation of permission data **10** (e.g. by any characteristic(s)) may be supported, for example for any subset of available in-process WDR data. Block **5924** may access any in-process WDR field, subfield(s), or associated in-process WDR data to make use of certain application interfaces accessible to the MS of FIG. **59** processing for read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to manage appropriate in-process WDR processing. If block **5926** determines the entry is not a WDR privilege, then processing continues to block **5930**. WDR privileges can be overall privileges, subordinate privileges, and/or privileges for any granulation of in-process related WDR data, perhaps using any characteristic(s) available from a derivative of the BNF grammar of FIGS. **30**A through **30**E.

If block **5930** determines the entry is a Situational Location privilege, then block **5932** will enable LBX features and functionality appropriately in context for the list parameter, and processing continues back to block **5908**. A Situational Location privilege may overlap with a WDR privilege since WDR fields are consulted for automated processing, however it may be useful to distinguish. Any granulation of permission data **10** (e.g. by any characteristic(s)) may be supported, for example for any subset of available in-process relevant WDR data. The term "situational location" is useful for describing location based conditions (e.g. as disclosed in Service delivered location dependent content of U.S. Pat. Nos. 6,456,234; 6,731,238; 7,187,997 (Johnson)). Block **5926** may access any in-process WDR field, subfield(s), or associated in-process WDR data for appropriate LBX processing involving read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to manage appropriate in-process WDR situational location processing. If block **5930** determines the entry is not a situational location privilege, then processing continues to block **5934**. Situation location privileges can be overall privileges, subordinate privileges, and/or privileges for any granulation of in-process related WDR data, perhaps using any characteristic(s) available from a derivative of the BNF grammar of FIGS. **30**A through **30**E.

If block **5934** determines the entry is a monitoring privilege, then block **5936** will enable LBX features and functionality appropriately in context for the list parameter, and processing continues back to block **5908**. A monitoring privilege governs monitoring any data of a MS for any reason (e.g. in charter conditions). Any granulation of permission data **10** (e.g. by any characteristic(s)) may be supported, for example for any subset of MS data. Block **5936** may access any MS data, or associated in-process WDR data for appropriate LBX processing involving read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to manage appropriate in-process WDR processing at the MS. If block **5936** determines the entry is not a monitoring privilege, then processing continues to block **5938**. Monitoring privileges can be overall privileges, subordinate privileges, and/or privileges for any granulation of MS data (MS of FIG. **59** processing or of the in-process WDR), perhaps using any characteristic(s) available from a derivative of the BNF grammar of FIGS. **30**A through **30**E.

If block **5938** determines the entry is a LBX privilege, then block **5940** will enable LBX features and functionality appropriately in context for the list parameter, and processing continues back to block **5908**. A LBX privilege governs LBX processing behavior at the MS of FIG. **59** processing. Other privileges so far discussed for FIG. **59** processing may overlap with an LBX privilege. Any granulation of permission data **10** (e.g. by any characteristic(s)) may be supported, for example for unique LBX processing at the MS of FIG. **59** processing. Block **5938** may access any MS data, or associated in-process WDR data for appropriate LBX processing involving read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to perform LBX processing at the MS. If block **5938** determines the entry is not a LBX privilege, then processing continues to block **5942**. LBX privileges can be overall privileges, subordinate privileges, and/or privileges for any granulation of MS data (MS of FIG. **59** processing or of the in-process WDR), perhaps using any characteristic(s) available from a derivative of the BNF grammar of FIGS. **30**A through **30**E.

If block **5942** determines the entry is a LBS privilege, then block **5944** will enable LBS features and functionality appropriately in context for the list parameter, and processing continues back to block **5908**. A LBS privilege governs LBS processing behavior at the MS of FIG. **59** processing. Other privileges so far discussed for FIG. **59** processing may overlap with an LBS privilege. Any granulation of permission data **10** (e.g. by any characteristic(s)) may be supported, for example for unique LBS processing at the MS of FIG. **59** processing. Block **5944** may access any MS data, or associated in-process WDR data for appropriate LBS processing involving read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to perform LBS processing at the MS, and perhaps cause processing at a connected LBS. If block **5942** determines the entry is not a LBS privilege, then processing continues to block **5946**. LBS privileges can be overall privileges, subordinate privileges, and/or privileges for any granulation of MS data (MS of FIG. **59** processing or of the in-process WDR), perhaps using any characteristic(s) available from a derivative of the BNF grammar of FIGS. **30**A through **30**E, and perhaps using any data or interface of a connected LBS.

Block **5946** is provided for processing completeness for handling appropriately (e.g. enable or disable MS processing) a privilege that some reader may not appreciate falling into one of the privilege classes of FIG. **59** processing. Block **5946** then continues to block **5908**. Referring back to block **5910**, if it is determined there are no more unprocessed entries remaining in the list parameter (i.e. **5810** of collection **5802** when FIG. **59** invoked from block **5734**), then the caller/invoker is returned to at block **5948**.

FIG. **59** may not require blocks **5904** and **5906** since a block **4466** embodiment may have already enforced what has been received and integrated at block **4470** to a proper set of collections **5802** and **5852**. In any case, the procedure of FIG. **59** is made complete having blocks **5904** and **5906** for various caller/invoker embodiments. Similarly, FIG. **59** also may not require blocks **5912** through **5916** since a block **4466** embodiment may have already enforced what has been received and integrated at block **4470** to a proper set of collections **5802**

and **5852**. The procedure of FIG. **59** is made complete by having blocks **5912** through **5916** for various caller/invoker embodiments.

In one embodiment, FIG. **59** uses the absence of certain privileges to enable or disable LBX features and functionality wherein block **5948**-A determines which privileges were not provided, block **5948**-B enables/disables LBX features and functionality in accordance with the lack of privileges, and block **5948**-C returns to the caller/invoker.

With reference back to FIG. **57**, block **5734** continues to block **5736**. Some embodiments of FIG. **57** blocks **5710**, **5714** **5718**, **5742**, **5750**, **5756**, etc may perform sorting for a best processing order (e.g. as provided to procedures of FIGS. **59** and **60**). Block **5736** performs actions in accordance with privileges of the PRIVS2ME list by invoking the do action procedure of FIG. **60** with the PRIVS2ME list, and the in-process WDR as parameters (preferably passed by pointer/reference).

With reference now to FIG. **60**, depicted is a flowchart for describing a preferred embodiment of a procedure for performing LBX actions in accordance with a certain type of permissions. Blocks **6012**, **6016**, **6020**, **6024**, **6028**, **6032**, **6036**, and **6038** perform actions for semantic privileges. Processing of block **5736** starts at block **6002** and continues to block **6004** where the permission type parameter passed (i.e. PRIVS2ME (**5810**) when invoked from block **5736**) is determined, and the in-process WDR may be accessed. The list parameter passed provides not only the appropriate list to FIG. **60** processing, but also which list configuration (**5810**, **5820**, **5830** or **5840**) has been passed for proper processing by FIG. **60**. There are potentially thousands of specific privileges that FIG. **60** can handle. Therefore, FIG. **60** processing is shown to generically handle different classes (categories) of privileges, namely privilege classes of: data send, impersonation, WDR processing, situational location, monitoring, LBX, LBS, and any others as handled by block **6038**. Privileges disclosed throughout the present disclosure fall into one of these classes handled by FIG. **60**.

Block **6004** continues to block **6006**. Block **6006** gets the next individual privilege entry (or the first entry upon first encounter of block **6006** for an invocation of FIG. **60**) from the list parameter and continues to block **6008**. Blocks **6006** through **6038** iterate all individual privileges associated with the list parameter of permissions **10** provided to block **6002**. If block **6008** determines there was an unprocessed privilege entry remaining in the list parameter (i.e. **5810** of collection **5802** when FIG. **60** invoked from block **5736**), then the entry gets processed starting with block **6010**.

If block **6010** determines the entry is a data send privilege, then block **6012** will perform any LBX actions in context for the list parameter (if any applicable), and processing continues back to block **6006**. A data send privilege may be one that is used at block **4466** and enforced at block **4470** for exactly what data can or cannot be received, or alternatively, block **6012** can perform actions for communicating data between MSs, or affecting data at MSs, for an appropriate local image of permissions **10** and/or charters **12**. Any granulation of permission data **10** or charter data **12** (e.g. by any characteristic(s)) may be supported. If block **6010** determines the list entry is not a data send privilege, processing continues to block **6014**.

If block **6014** determines the entry is an impersonation privilege, then block **6016** will perform any LBX actions in context for the list parameter (if any applicable), and processing continues back to block **6006**. Block **6016** may access security, or certain application interfaces accessible to the MS of FIG. **60** processing for read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to manage associated identity impersonation at the MS. If block **6014** determines the entry is not an impersonation privilege, then processing continues to block **6018**.

If block **6018** determines the entry is a WDR privilege, then block **6020** will perform any LBX actions in context for the list parameter (if any applicable), and processing continues back to block **6006**. Block **6020** may access any in-process WDR field, subfield(s), or associated in-process WDR data to make use of certain application interfaces accessible to the MS of FIG. **60** processing for read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to manage appropriate in-process WDR processing. If block **6020** determines the entry is not a WDR privilege, then processing continues to block **6022**.

If block **6022** determines the entry is a Situational Location privilege, then block **6024** will perform any LBX actions in context for the list parameter (if any applicable), and processing continues back to block **6006**. Block **6024** may access any in-process WDR field, subfield(s), or associated in-process WDR data for appropriate LBX processing involving read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to manage appropriate in-process WDR situational location processing. If block **6022** determines the entry is not a situational location privilege, then processing continues to block **6026**

If block **6026** determines the entry is a monitoring privilege, then block **6028** will perform any LBX actions in context for the list parameter (if any applicable), and processing continues back to block **6006**. Block **6028** may access any MS data, or associated in-process WDR data for appropriate LBX processing involving read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to manage appropriate in-process WDR processing at the MS. If block **6026** determines the entry is not a monitoring privilege, then processing continues to block **6030**.

If block **6030** determines the entry is a LBX privilege, then block **6032** will perform any LBX actions in context for the list parameter (if any applicable), and processing continues back to block **6006**. Block **6032** may access any MS data, or associated in-process WDR data for appropriate LBX processing involving read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to perform LBX processing at the MS. If block **6030** determines the entry is not a LBX privilege, then processing continues to block **6034**.

If block **6034** determines the entry is a LBS privilege, then block **6036** will perform any LBS actions in context for the list parameter, and processing continues back to block **6006**. Block **6036** may access any MS data, or associated in-process WDR data for appropriate LBS processing involving read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to perform LBS processing at the MS, and perhaps cause processing at a connected LBS. If block **6034** determines the entry is not a LBS privilege, then processing continues to block **6038**.

Block **6038** is provided for processing completeness for handling appropriately (e.g. performing any LBX actions in context for the list parameter (if any applicable) a privilege that some reader may not appreciate falling into one of the privilege classes of FIG. **60** processing. Block **6038** then continues to block **6006**. Referring back to block **6008**, if it is

determined there are no more unprocessed entries remaining in the list parameter (i.e. **5810** of collection **5802** when FIG. **60** invoked from block **5736**), then the caller/invoker is returned to at block **6040**.

In one embodiment, FIG. **60** uses the absence of certain privileges to perform LBX actions in context for the list parameter wherein block **6040**-A determines which privileges were not provided, block **6040**-B performs LBX actions in context for the lack of privileges, and block **6040**-C returns to the caller/invoker.

FIG. **60** processing causes application types of actions according to privileges set. Such application types of actions are preferably caused using APIs, callback functions, or other interfaces so as to isolate FIG. **60** LBX processing from applications that are integrated with it. This prevents application "know-how" from being part of the LBX processing (e.g. software) built for MSs. FIG. **60** preferably invokes the "know-how" through an appropriate interface (software or hardware). In one preferred embodiment, participating applications register themselves as processing particular atomic privileges so that FIG. **60** invokes the interface with the privilege, its setting, and perhaps useful environmental data of interest. The application itself can then optimally process the privilege for an appropriate application action. Invocation of the application interface may be thread oriented so as to not wait for a return, or may be synchronous for waiting for a return (or return code). In one preferred embodiment, the PRR **5300** is modified for further containing a privilege join field **5300***j* for joining to a new Application Privileges Reference (APR) table containing all privileges which are relevant for the application described by the PRR **5300**. This provides the guide of all privileges which are applicable to an application, and which are to cause invocation of the interface(s) of the application. A PRR **5300** is to be extended with new data in at least one field **5300***k* which contains interface directions for how to invoke the application with the privilege for processing (e.g. through a Dynamic Link Library (DLL), or script, interface). Preferably, a single API or invocation is used for all privileges to a particular application and the burden of conditional processing paths is put on the application in that one interface. An alternate embodiment could allow multiple interfaces to be plugged in: one for each of a plurality of classes, or categories, of privileges so that the burden of unique processing paths, depending on a privilege, is reduced for one application. In any embodiment, it is preferable to minimize linkage execution time between LBX processing and an application which is plugged in. Linkage time can be reduced by:

1) Performing appropriate and directed executable linkage as indicated by the PRR at initialization time of block **1240**;
2) Performing loading into executable memory of needed dynamically linked executables (e.g. DLL) as indicated by the PRR at initialization time of block **1240** wherein the PRR provides link library information for resolving linkage; and/or
3) Validating presence of, or performing loading of, the executables/script/etc in an appropriate manner at an appropriate initialization time.

Note that atomic command processing solves performance issues by providing a tightly linked executable environment while providing methods for customized processing. Many applications may be invoked for the same privilege (i.e. blocks **6012**, **6016**, **6020**, **6024**, **6028**, **6032**, **6036** and/or **6038** can certainly invoke multiple applications (i.e. cause multiple actions) for a single privilege), depending on what is found in the APR table. Of course, integrated application action processing can be built with LBX software so that the MS applications are tightly integrated with the LBX processing. Generally, FIG. **60** includes appropriate processing of applications while FIG. **59** affects data which can be accessed (e.g. polled) by applications.

With reference back to FIG. **57**, block **5736** continues to block **5738**. Block **5738** performs actions in accordance with privileges of the PRIVS2WDR list by invoking the do action procedure of FIG. **60** with the PRIVS2WDR list, and the in-process WDR as parameters (preferably passed by pointer/reference), and then continues to block **5740**. FIG. **60** processing is analogously as described above except in context for the PRIVS2WDR (**5820**) list and for the in-process WDR of FIG. **57** processing relative the PRIVS2WDR list. One embodiment may incorporate a block **5737** (block **5736** continues to **5737** which continues to block **5738**) for invoking FIG. **59** processing with PRIVS2WDR. Generally, privilege configurations **5820** involve actions for the benefit of the WDR originator.

Block **5740** processing merges the MYCHARTERS and CHARTERS2ME lists into a CHARTERS2DO list, and continues to block **5742** for eliminating inappropriate charters that may exist in the CHARTERS2DO list. Block **5742** additionally eliminates charters with a TimeSpec qualifier invalid for the time of FIG. **57** processing (an alternate embodiment can enforce TimeSpec interpretation at block **5744**). If all actions, or any condition, term, expression, or entire charter itself has a TimeSpec outside of the time of FIG. **57** processing, then preferably the entire charter is eliminated. Action(s) are removed from a charter which remains in effect if action(s) for a charter have an invalid TimeSpec for the time of FIG. **57** processing, in which case any remaining actions with no TimeSpec or a valid TimeSpec are preserved for the effective charter. If all charter actions are invalid per TimeSpec, then the charter is completely eliminated. Thereafter, block **5744** performs charter actions in accordance with conditions of charters of the CHARTERS2DO list (see FIG. **61**), and processing then terminates at block **5746**.

Block **5742** can eliminate charters which are irrelevant for processing, for example depending upon the type of in-process WDR. For a maintained WDR, inappropriate charters may be those which do not have a maintained condition specification (i.e. _fldname). For an inbound WDR, inappropriate charters may be those which do not have an in-bound condition specification (i.e. _I_fldname). For an outbound WDR, inappropriate charters may be those which do not have an out-bound condition specification (i.e. _I_fldname). The context of WITS processing (mWITS, iWITS, oWITS) may be used at block **5742** for eliminating inappropriate charters.

With reference back to block **5732**, if it is determined that this MS should not process (see) the WDR in-process, processing continues to block **5746** where FIG. **57** processing is terminated, and the processing host of FIG. **57** (i.e. FIGS. **2**F **20**, **21**, **25**) appropriately ignores the WDR.

With reference back to block **5706**, if it is determined that the WDR identity matches the MS of FIG. **57** processing, processing continues to block **5748**. Block **5706** continues to block **5748** when a) the in-process WDR is from this MS and is being maintained at the MS of FIG. **57** processing (i.e. FIG. **57**=mWITS); or b) the in-process WDR is outbound from this MS (i.e. FIG. **57**=oWITS). Block **5748** forms a PRIVS2OTHERS list of configurations **5830** and continues to block **5750** for eliminating duplicates that may be found. Block **5748** may collapse grant hierarchies to form the list. Duplicates may occur for privileges which include the duplicated privileges (i.e. subordinate privileges) as described above. Block **5750** additionally eliminates duplicates that

may exist for permission embodiments wherein a privilege can enable or disable a feature. In a present disclosure embodiment wherein a privilege can enable, and a privilege can disable the same feature or functionality, there is preferably a tie breaker of disabling the feature (i.e. disabling wins). In an alternate embodiment, enabling may break a tie of ambiguity. Block **5750** further eliminates privileges that have a MSRelevance qualifier indicating the MS of FIG. **57** processing is not supported for the particular privilege, and also eliminates privileges with a TimeSpec qualifier invalid for the time of FIG. **57** processing (an alternate embodiment can enforce TimeSpec interpretation at block **5758** (i.e. in FIG. **60** processing)). Thereafter, block **5752** forms a MYCHARTERS list of configurations **5870** and preferably eliminates variables by instantiating/elaborating at points where they are referenced. Then, block **5754** forms a CHARTERS2ME list of configurations **5890** and preferably eliminates variables by instantiating/elaborating at points where they are referenced. Then, block **5756** eliminates those charters which are not privileged. In some embodiments, block **5756** is not necessary (**5754** continues to **5758**) because un-privileged charters will not be permitted to be present at the MS of FIG. **57** processing. Nevertheless, block **5756** removes from the CHARTERS2ME list all charters which do not have a privilege granted by the MS (the MS user) of FIG. **57** processing to the creator of the charter, for permitting the charter to be enabled (as described above for block **5718**). In any embodiments, block **5756** ensures no charters from other users are considered active unless appropriately privileged. Thereafter, block **5758** performs actions in accordance with privileges of the PRIVS2OTHERS list by invoking the do action procedure of FIG. **60** with the PRIVS2ME list, and the in-process WDR as parameters (preferably passed by pointer/reference), and then continues to block **5740** which has already been described. FIG. **60** processing is the same as described above except in context for the PRIVS2OTHERS (**5830**) and for the in-process WDR of FIG. **57** processing relative the PRIVSOTHERS list. Of course the context of blocks **5748** through **5758** are processed for in-process WDRs which are: a) maintained to the MS of FIG. **57** for the whereabouts of the MS of FIG. **57** processing; or b) outbound from the MS of FIG. **57** processing (e.g. an outbound WDR describing whereabouts of the MS of FIG. **57** processing). One embodiment may incorporate a block **5757** (block **5756** continues to **5757** which continues to block **5758**) for invoking FIG. **59** processing with PRIVS2OTHERS. Generally, privilege configurations **5830** involve actions for the benefit of others (i.e. other than this MS).

When considering the terminology "incoming" as used for FIGS. **49**A and **49**B, a WDR in-process at this MS (the MS of FIG. **57** processing) which was originated by this MS with an identity for this MS uses: a) this MS charters (**5870** confirmed by **4962** bullet **2** part **1**, **4988** bullet **2** part **1**, **4922**, **4948**); b) others' charters per this MS (or this MS user) privileges to them (**5890** confirmed by **4966** bullet **3**, **4964** bullet **2**, **4986** bullet **3**, **4984** bullet **2**, **4924**, **4946**); and c) this MS (or this MS user) privileges to others (**5830** confirmed by **4944** bullet **4**, **4924** bullet **4**, **4946** bullet **4**, **4926** bullet **4**). An alternate embodiment additionally uses d) others' privileges to this MS (or this MS user) (**5840**), for example to determine how nearby they are at outbound WDR time or at the time of maintaining the MS's own whereabouts. This alternate embodiment would cause FIG. **57** to include: a new block **5760** for forming a PRIVS2ME list of privileges **5840**; a new block **5762** for eliminating duplicates, MSRelevance rejects and invalid TimeSpec entries; a new block **5764** for enabling features an functionality in accordance with the PRIVS2ME list of block **5760** by invoking the enable features and functionality procedure of FIG. **59** with PRIVS2ME as a parameter (FIG. **59** processing analogous to as described above except for PRIVS2ME); and a new block **5766** for performing actions in accordance with PRIVS2ME by invoking the do action procedure of FIG. **60** with PRIVS2ME as a parameter (FIG. **60** processing analogous to as described above except for PRIVS2ME). Such an embodiment would cause block **5758** to continue to block **5760** which continues to block **5762** which continues to block **5764** which continues to block **5766** which then continues to block **5740**.

When considering the terminology "incoming" as used for FIGS. **49**A and **49**B, a WDR in-process at this MS (the MS of FIG. **57** processing) which was originated by a remote MS with an identity different than this MS uses: e) this MS charters per other's privileges to this MS (or this MS user) (**5870** confirmed by **4962** bullet **2** part **2**, **4988** bullet **2** part **2**, **4926**, **4944**, **4924** bullet **2**); f) others' charters per this MS (or this MS user) privileges to them (**5860** confirmed by **4966** bullet **2**, **4964** bullet **3**, **4986** bullet **2**, **4984** bullet **3**, **4924**, **4946**); g) this MS (or this MS user) privileges to others (**5820** confirmed by **4944** bullet **3**, **4924** bullet **3**, **4946** bullet **3**, **4926** bullet **3**); and h) others' privileges to this MS (or this MS user) (**5810** confirmed by **4926** bullet **2**, **4944** bullet **2**, **4946** bullet **2**, **4924** bullet **2**). An alternate embodiment additionally uses i) others' charters per this MS (or this MS user) privileges to them (**5890**); and/or j) this MS (or this MS user) privileges to others (**5830**); and/or k) others' privileges to this MS (or this MS user) (**5840**). This alternate embodiment would cause FIG. **57** to alter block **5716** to further include charters **5890**, alter block **5708** to further include privileges **5840**, include a new block **5722** for forming a PRIVS2OTHERS list of privileges **5830**, new block **5724** for eliminating duplicates, new block **5726** for enabling features an functionality in accordance with the PRIVS2OTHERS list of block **5722**, new block **5728** for enabling features an functionality in accordance with the modified PRIVS2ME list of block **5708**, and new block **5730** for performing actions in accordance with the modified PRIVS2ME (i.e. block **5720** continues to block **5722** which continues to block **5724** which continues to block **5726** which continues to block **5728** which continues to block **5730** which then continues to block **5732**). Also, blocks **5742** and **5744** would appropriately handle new charters of altered block **5716**. Such an embodiment would cause new blocks **5726**, **5728** and **5730** to invoke the applicable procedure (FIG. **59** or FIG. **60**) with analogous processing as described above except in context for the parameter passed.

In some FIG. **57** embodiments, blocks **5708** and/or **5716** and/or **5754** and/or relevant alternate embodiment blocks discussed are remotely accessed by communicating with the MS having the identity determined at block **5704** for the WDR in-process. The preferred embodiment is as disclosed for maintaining data local to the MS for processing there. In other embodiments, there are separate flowcharts (e.g. FIGS. **57**A, **57**B and **57**C) for each variety of handling in-process WDRs (e.g. mWITS, iWITS, oWITS processing).

Various FIG. **57** embodiments' processing will invoke the procedure of FIG. **59** with appropriate parameters (i.e. lists for **5810** and/or **5820** and/or **5830** and/or **5840**) so that any category subset of the permission data collection **5802** (i.e. **5810** and/or **5820** and/or **5830** and/or **5840**) is used to enable appropriate LBX features and functionality according to the WDR causing execution of FIG. **57** processing. For example, privileges between the MS of FIG. **57** processing and an identity other than the WDR causing FIG. **57** processing may be used (e.g. relevant MS third party notification, features, functionality, or processing as defined by related privileges).

Various FIG. **57** embodiments' processing will invoke the procedure of FIG. **60** with appropriate parameters (i.e. lists for **5860** and/or **5870** and/or **5880** and/or **5890**) so that any category subset of the charter data collection **5852** (i.e. **5860** and/or **5870** and/or **5880** and/or **5890**) is used to perform LBX actions according to the WDR causing execution of FIG. **57** processing. For example, charters between the MS of FIG. **57** processing and an identity other than the WDR causing FIG. **57** processing may be used (e.g. relevant MS third party charters as defined by related privileges).

FIG. **57** determines which privileges and charters are relevant to the WDR in process, regardless of where the WDR originated. The WDR identity checked at block **5706** can take on various embodiments so that the BNF grammar of FIGS. **30**A through **30**E are fully exploited. Preferably, the identities associated with "this MS" and the WDR in process are usable as is, however while there are specific embodiments implementing the different identifier varieties, there may also be a translation or lookup performed at block **5704** to ensure a proper compare at block **5706**. The identities of "this MS" and the WDR identity (e.g. field **1100***a*) may be translated prior to performing a compare. For example, a user identifier maintained to the user configurations (permissions/charters) may be "looked up" using the MS identifiers involved ("this MS" and WDR MS ID) in order to perform a proper compare at block **5706**. Some embodiments may maintain a separate identifier mapping table local to the MS, accessed from a remote MS when needed, accessed from a connected service, or accessed as is appropriate to resolve the source identifiers with the identifiers for comparing at block **5706**. Thus, permissions and/or charters can grant from one identity to another wherein identities of the configuration are associated directly (i.e. useable as is) or indirectly (i.e. mapped) to the actual identities of the user(s), the MS(s), the group(s), etc involved in the configuration.

Preferably, statistics are maintained by WITS processing for each reasonable data worthy of tracking from standpoints of user reporting, automated performance fine tuning (e.g. thread throttling), automated adjusted processing, and monitoring of overall system processing. In fact, every processing block of FIG. **57** can have a plurality of statistics to be maintained.

FIG. **61** depicts a flowchart for describing a preferred embodiment of performing processing in accordance with configured charters, as described by block **5744**. The CHARTERS2DO list from FIG. **57** is processed by FIG. **61**. FIG. **61** (and/or FIG. **57** (e.g. blocks **5718**/**5756**)) is responsible for processing grammar specification privileges. Block **5744** processing begins at block **6102** and continues to block **6104**. Block **6104** gets the next charter (or first charter on first encounter to block **6104** from block **6102**) from the CHARTERS2DO list and continues to block **6106** to check if all charters have already been processed from the list. Block **6104** begins an iterative loop (blocks **6104** through **6162**) for processing all charters (if any) from the CHARTERS2DO list.

If block **6106** determines there is a charter to process, then processing continues to block **6108** for instantiating any variables that may be referenced in the charter, and then continues to block **6110**. Charter parts are scanned for referenced variables and they are instantiated so that the charter is intact without a variable reference. The charter internalized form may be modified to accommodate instantiation(s). FIG. **57** may have already instantiated variables for charter elimination processing. Block **6108** is typically not required since the variables were likely already instantiated when internalized to a preferred embodiment CHARTERS2DO processable form, and also processed by previous blocks of FIG. **57** processing. Nevertheless, block **6108** is present to cover other embodiments, and to handle any instantiations which were not already necessary. In some embodiments, block **6108** is not required since variable instantiations can occur as needed when processing the individual charter parts during subsequent blocks of FIG. **61** processing. Block **6106** would continue to block **6110** when a block **6108** is not required.

Block **6110** begins an iterative loop (blocks **6110** through **6118**) for processing all special terms from the current charter expression. Block **6110** gets the next (or first) special term (if any) from the charter expression and continues to block **6112**. A special term is a BNF grammar WDRTerm, AppTerm, or atomic term. If block **6112** determines a special term was found for processing from the expression, then block **6114** accesses privileges to ensure the special term is privileged for use. Appropriate permissions **5802** are accessed in this applicable context of FIG. **57** processing. Block **6114** then continues to block **6116**. Blocks **6114** and **6116** may not be required since unprivileged charters were already eliminated in previous blocks of FIG. **57** processing (e.g. see blocks **5718** and **5756**). Nevertheless, blocks **6114** and **6116** are shown to cover other embodiments, and to ensure unprivileged charters are treated ineffective. Depending on an embodiment, blocks **5718** and **5756** may only perform obvious eliminations. In other embodiments, there may be no blocks **5718** or **5756** so that charter part processing occurs only in one place (i.e. FIG. **61**) to achieve better MS performance by preventing more than one scan over charter data. In another embodiment, blocks **6114** and **6116** are not required since all charter eliminations based on privileges already occurred at the previous blocks of FIG. **57** processing. Block **6112** can continue to block **6118** when blocks **6114** and **6116** are not required.

If block **6116** determines the special term is privileged for use (e.g. explicit privilege, or lack of a privilege denying use, depending on privilege deployment embodiments), then block **6118** appropriately accesses the special term data source and replaces the expression referenced special term with the corresponding value. Block **6118** accesses special term data dynamically so that the terms reflect values at the time of block **6118** processing. Block **6118** continues back to block **6110**. A WDRTerm is accessed from the in-process WDR to FIG. **57** processing. An AppTerm is an anticipated registered application variable accessed by a well known name, typically with semaphore control since an asynchronous application thread is writing to the variable. An atomic term will cause access to WDR data at queue **22** or LBX history **30**, application status for applications in use at the MS of FIG. **57** processing, system date/time, the MS ID of the MS of FIG. **57** processing, or other appropriate data source.

Referring back to block **6116**, if it is determined that the special term of the charter expression is not privileged, then block **6120** logs an appropriate error (e.g. to LBX history **30**) and processing continues back to block **6104** for the next charter. An alternate block **6120** may alert the MS user, and in some cases require the user to acknowledge the error before continuing back to block **6104**. So, the preferred embodiment of charter processing eliminates a charter from being processed if any single part of the charter expression is not privileged.

Referring back to block **6112**, if it is determined there are no special terms in the expression remaining to process (or there were none in the expression), then block **6122** evaluates the expression to a Boolean True or False result using well known processing for a stack based parser for expression evaluation (e.g. See well known compiler/interpreter development techniques (e.g. "Algorithms+Data

Structures=Programs" by Nicklaus Wirth published by Prentice-Hall, Inc. 1976)). Block **6122** implements atomic operators using the WDR queue **22**, most recent WDR for this MS, LBX history **30**, or other suitable MS data. Any Invocation is also invoked for resulting to a True or False wherein a default is enforced upon no return code, or no suitable return code, returned. Invocation parameters that had special terms would have been already been updated by block **6118** to eliminate special terms prior to invocation. Thereafter, if block **6124** determines the expression evaluated to False, then processing continues back to block **6104** for the next charter (i.e. expression=False implies to prevent (not cause) the action(s) of the charter). If block **6124** determines the expression evaluated to True, then processing continues to block **6126**.

Block **6126** begins an iterative loop (blocks **6126** through **6162**) for processing all actions from the current charter. Block **6126** gets the next (or first) action (if any) from the charter and continues to block **6128**. There should be at least one action in a charter provided to FIG. **61** processing since the preferred embodiment of FIG. **57** processing will have eliminated any placeholder charters without an action specified (e.g. charters with no actions preferably eliminated at blocks **5740** as part of the merge process, at block **5742**, or as part of previous FIG. **57** processing to form privileged charter lists). If block **6128** determines an unprocessed action was found for processing, then block **6130** initializes a REMOTE variable to No. Thereafter, if it is determined at block **6132** that the action has a BNF grammar Host specification, then block **6134** accesses privileges and block **6136** checks if the action is privileged for being executed at the Host specified. The appropriate permissions **5802** are accessed at block **6134** in this applicable context of FIG. **57** processing. If block **6136** determines the action is privileged for running at the Host, then block **6138** sets the REMOTE variable to the Host specified and processing continues to block **6140**. If block **6136** determines the action is not privileged for running at the Host, then processing continues to block **6120** for error processing already described above. If block **6132** determines there was no Host specified for the action, processing continues directly to block **6140**. Blocks **6134** and **6136** may not be required since unprivileged charters were already eliminated in previous blocks of FIG. **57** processing (e.g. see blocks **5718** and **5756**). Nevertheless, blocks **6134** and **6136** are shown to cover other embodiments, and to ensure unprivileged charters are treated ineffective. Depending on an embodiment, blocks **5718** and **5756** may only perform obvious eliminations. In other embodiments, there may be no blocks **5718** or **5756** so that charter part processing occurs only in one place (i.e. FIG. **61**) to achieve better MS performance by preventing more than one scan over charter data. In another embodiment, blocks **6134** and **6136** are not required since all charter eliminations based on privileges already occurred at the previous blocks of FIG. **57** processing. Block **6132** can continue to block **6138** when blocks **6134** and **6136** are not required and a Host was specified with the action.

Block **6140** accesses appropriate permissions **5802** in this applicable context of FIG. **57** processing for ensuring the command and operand are appropriately privileged. Thereafter, if block **6142** determines that the action's command and operand are not privileged, then processing continues to block **6120** for error processing already described. If block **6142** determines the action's command and operand are to be effective, then processing continues to block **6144**. Blocks **6140** and **6142** may not be required since unprivileged charters were already eliminated in previous blocks of FIG. **57** processing (e.g. see blocks **5718** and **5756**). Nevertheless, blocks **6140** and **6142** are shown to cover other embodiments, and to ensure unprivileged charters are treated ineffective. Depending on an embodiment, blocks **5718** and **5756** may only perform obvious eliminations. In other embodiments, there may be no blocks **5718** or **5756** so that charter part processing occurs only in one place (i.e. FIG. **61**) to achieve better MS performance by preventing more than one scan over charter data. In another embodiment, blocks **6140** and **6142** are not required since all charter eliminations based on privileges already occurred at the previous blocks of FIG. **57** processing. Block **6138**, and the No condition of block **6132**, would continue to block **6144** when blocks **6140** and **6142** are not required.

Block **6144** begins an iterative loop (blocks **6144** through **6152**) for processing all parameter special terms of the current charter. Block **6144** gets the next (or first) parameter special term (if any) and continues to block **6146**. A special term is a BNF grammar WDRTerm, AppTerm, or atomic term (as described above). If block **6146** determines a special term was found for processing from the parameter list, then block **6148** accesses privileges to ensure the special term is privileged for use. The appropriate permissions **5802** are accessed in this applicable context of FIG. **57** processing. Block **6148** then continues to block **6150**. Blocks **6148** and **6150** may not be required since unprivileged charters were already eliminated in previous blocks of FIG. **57** processing (e.g. see blocks **5718** and **5756**). Nevertheless, blocks **6148** and **6150** are shown to cover other embodiments, and to ensure unprivileged charters are treated ineffective. Depending on an embodiment, blocks **5718** and **5756** may only perform obvious eliminations. In other embodiments, there may be no blocks **5718** or **5756** so that charter part processing occurs only in one place (i.e. FIG. **61**) to achieve better MS performance by preventing more than one scan over charter data. In another embodiment, blocks **6148** and **6150** are not required since all charter eliminations based on privileges already occurred at the previous blocks of FIG. **57** processing. Block **6146** can continue to block **6152** when blocks **6148** and **6150** are not required.

If block **6150** determines the special term is privileged for use (e.g. explicit privilege, or lack of a privilege denying use, depending on privilege deployment embodiments), then block **6152** appropriately accesses the special term data source and replaces the parameter referenced special term with the corresponding value. Block **6152** accesses special term data dynamically so that the terms reflect values at the time of FIG. **61** block **6152** processing. Block **6152** continues back to block **6144**. A WDRTerm, AppTerm, and atomic term are accessed in a manner analogous to accessing them at block **6118**.

Referring back to block **6150**, if it is determined that the special term of the parameter list is not privileged, then processing continues to block **6120** for error processing already described. Referring back to block **6146**, if it is determined there are no special terms in the parameter list remaining to process (or there were none), then block **6154** evaluates each and every parameter expression to a corresponding value using well known processing for a stack based parser for expression evaluation (e.g. See well known compiler/interpreter development techniques (e.g. "Algorithms+Data Structures=Programs" by Nicklaus Wirth published by Prentice-Hall, Inc. 1976)). Block **6154** implements the atomic operators using the WDR queue **22**, most recent WDR for this MS, LBX history **30**, or other suitable MS data. Any Invocation is also invoked for resulting to Data or Value wherein a default is enforced upon no returned data. Invocation parameters that had special terms would have been updated at block **6152** to eliminate special terms prior to invocation. Block