6154 ensures each parameter is in a ready to use form to be processed with the command and operand. Each parameter results in embodiments of a data value, a data value resulting from an expression, a data reference (e.g. pointer), or other embodiments well known in the art of passing parameters (arguments) to a function, procedure, or script for processing. Thereafter, if block 6156 determines the REMOTE variable is set to No (i.e. "No" equals a value distinguishable from any Host specification for having the meaning of "No Host Specification"), then processing continues to block 6158 where the ExecuteAction procedure of FIG. 62 is invoked with the command, operand and parameters of the action in process. Upon return from the procedure of FIG. 62, processing continues back to block 6126 for any remaining charter actions. If block 6156 determines the REMOTE variable is set to a Host for running the action, then processing continues to block 6160 for preparing send data procedure parameters for performing a remote action (of the command, operand and parameters), and then invoking the send data procedure of FIG. 75A for performing the action at the remote MS (also see FIG. 75B). Processing then continues back to block 6126. An alternate embodiment will loop on multiple BNF grammar Host specifications for multiple invocations of the send data procedure (i.e. when multiple Host specifications are supported). Another embodiment to FIG. 61 processing permits multiple actions with a single Host specification.

Referring back to block 6128, if it is determined all current charter actions are processed, then processing continues to block 6104 for any next charter to process. Referring back to block 6106, if it is determined all charters have been processed, processing terminates at block 6164.

Depending on various embodiments, there may be obvious error handling in FIG. 61 charter parsing. Preferably, the charters were reasonably validated prior to being configured and/or previously processed/parsed (e.g. FIG. 57 processing). Also, TimeSpec and/or MSRelevance information may be used in FIG. 61 so that charter part processing occurs only in one place (i.e. FIG. 61 rather than FIG. 57) to achieve better MS performance by preventing more than one scan over charter data. Some embodiments of FIG. 61 may be the single place where charters are eliminated based on privileges, TimeSpecs, MSRelevance, or any other criteria discussed with FIG. 57 for charter elimination to improve performance (i.e. a single charter parse when needed). Third party MSs (i.e. those that are not represented by the in-process WDR and the MS of FIG. 57 processing) can be affected by charter actions (e.g. via Host specification, privileged action, privileged feature, etc).

Preferably, statistics are maintained throughout FIG. 61 processing for how charters were processed, which charters became effective, why they became effective, which commands were processed (e.g. invocation of FIG. 62), etc.

With reference now to FIG. 75A, depicted is a flowchart for describing a preferred embodiment of a procedure for sending data to a remote MS, for example to perform a remote action as invoked from block 6162. FIG. 75A is preferably of linkable PIP code 6. The purpose is for the MS of FIG. 75A processing (e.g. a first, or sending, MS) to transmit data to other MSs (e.g. at least a second, or receiving, MS), for example an action (command, operand, and any parameter(s)), or specific processing for a particular command (e.g. Send atomic command). 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. 75A (e.g. block 6162), there can be intelligence to drive each channel for broadcasting on multiple channels, either by multiple send threads for FIG. 75A processing, FIG. 75A loop processing on a channel list, and/or passing channel information to send processing feeding from queue 24. If FIG. 75A 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. 13A through 13C, as well as long range exemplifications shown in FIGS. 50A through 50C, respectively. Processing begins at block 7502, continues to block 7504 where the caller parameter(s) passed to FIG. 75A processing (e.g. action for remote execution, or command for remote execution) are used for sending at least one data packet containing properly formatted data for sending, and for being properly received and interpreted. Block 7504 may reformat parameters into a suitable data packet(s) format so the receiving MS can process appropriately (see FIG. 75B). Depending on the present disclosure embodiment, any reasonable supported identity (ID/IDType) is a valid target (e.g. as derived from a recipient or system parameter). Thereafter, block 7506 waits for an acknowledgement from the receiving MS if the communication embodiment in use utilizes that methodology. In one embodiment, the send data packet is an unreliable datagram that will most likely be received by the target MS. In another embodiment, the send data packet is reliably transported data which requires an acknowledgement that it was received in good order. In any case, block 7506 continues to block 7508.

Block 7504 formats the data for sending in accordance with the specified delivery method, along with necessary packet information (e.g. source identity, wrapper data, etc), and sends data appropriately. For a broadcast send, block 7504 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 7506. The broadcast is for reception by data processing systems (e.g. MSs) in the vicinity of FIGS. 13A through 13C, as further explained by FIGS. 50A through 50C which includes potentially any distance. The targeted MS should recognize that the data is meant for it and receives it. For a targeted send, block 7504 formats the data intended for recognition by the receiving target. 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 7504 processing, will place 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. 75A sends/broadcasts new data 1302.

Block 7506 waits for a synchronous acknowledgement if applicable to the send of block 7504 until either receiving one or timing out. Block 7506 will not wait if no ack/response is anticipated, in which case block 7506 sets status for block 7508 to "got it". 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. Thereafter, if block 7508 determines an applicable ack/response was received (i.e. data successfully sent/

received), or none was anticipated (i.e. assume got it), then processing continues to block **7510** for potentially processing the response. Block **7510** will process the response if it was anticipated for being received as determined by data sent at block **7504**. Thereafter, block **7512** performs logging for success (e.g. to LBX History **30**). If block **7508** determines an anticipated ack was not received, then block **7512** logs the attempt (e.g. to LBX history **30**). An alternate embodiment to block **7514** will log an error and may require a user action to continue processing so a user is confirmed to have seen the error. Both blocks **7512** and **7514** continue to block **7516** where the caller (invoker) is returned to for continued processing (e.g. back to block **6162**).

With reference now to FIG. **75**B, depicted is a flowchart for describing a preferred embodiment of processing for receiving execution data from another MS, for example action data for execution, or processing of a particular atomic command for execution. FIG. **75**B processing describes a Receive Execution Data (RxED) process worker thread, and is of PIP code **6**. There may be many worker threads for the RxED process, just as described for a **19***xx* process. The receive execution data (RxED) 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 RxED thread(s) stay blocked on the receive queue until data is received, and a RxED 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 RxED process. For example, the RxED process is initialized as part of the enumerated set at blocks **1226** (e.g. preferably next to last member of set) and **2806** (e.g. preferably second member of set) for similar architecture **1900** processing. Receive processing identifies targeted/broadcasted data destined for the MS of FIG. **75**B processing. An appropriate data format is used, for example using X.409 encoding of FIGS. **33**A through **33**C for some subset of data packet(s) received wherein RxED thread(s) purpose is for the MS of FIG. **75**B processing to respond to incoming data. It is recommended that validity criteria set at block **1444** for RxED-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 RxED process, there can be a RxED worker thread per channel to handle receiving on multiple channels simultaneously. If RxED thread(s) do not receive directly from the channel, the preferred embodiment of FIG. **75**B would not need to convey channel information to RxED thread(s) waiting on queue **24** anyway. Embodiments could allow specification/configuration of many RxED thread(s) per channel.

A RxED thread processing begins at block **7552**, continues to block **7554** where the process worker thread count RxED-Ct is accessed and incremented by 1 (using appropriate semaphore access (e.g. RxED-Sem)), and continues to block **7556** for retrieving from queue **26** sent data (using interface like interface **1948**), perhaps a special termination request entry, and only continues to block **7558** when a record of data (e.g. action for remote execution, particular atomic command, or termination record) is retrieved. In one embodiment, receive processing deposits data as record(s) to queue **26**. 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. **75**B processing.

Block **7556** stays blocked on retrieving from queue **26** until data is retrieved, in which case processing continues to block **7558**. If block **7558** determines a special entry indicating to terminate was not found in queue **26**, processing continues to block **7560**. There are various embodiments for RxED thread(s), 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, **26**C and **26**D, 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**D and **26**E for separate processing of incoming remote action and send command data. In another embodiment, thread(s) **1912** are modified with logic of RxED thread(s) to handle remote actions and send command data requests, since thread(s) **1912** are listening for queue **26** data anyway. In yet another embodiment, there are distinct threads and/or distinct queues for processing each kind of an atomic command to FIG. **75**B processing (i.e. as processed by blocks **7578** through **7584**).

Block **7560** validates incoming data for this targeted MS before continuing to block **7562**. 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 **7558** can continue directly to block **7564** (no block **7560** and block **7562** required)). If block **7562** determines the data is valid for processing, then block **7564** checks the data for its purpose (remote action or particular command). If block **7564** determines the data received is for processing a remote action, then block **7566** accesses source information, the command, the operand, and parameters from the data received. Thereafter, block **7568** accesses privileges for each of the remote action parts (command, operand, parameters) to ensure the source has proper privileges for running the action at the MS of FIG. **75**B processing. Depending on embodiments, block **7568** may include evaluating the action for elaborating special terms and/or expressions as described for FIG. **61** (blocks **6140** through **6154**), although the preferred embodiment preferably already did that prior to transmitting the remote action for execution (e.g. remote action already underwent detailed privilege assessment). However, in some embodiments where privileges are only maintained locally, the action processing of FIG. **61** processing would be required at block **7568** to check privileges where appropriate in processing the action. In such embodiments, FIG. **61** would process local actions as disclosed, but would not process actions known to be for remote execution (i.e. Host specification) since a FIG. **75**B embodiment would include FIG. **61** processing for performing privilege check processing to determine that sufficient privileges are granted. Thus, depending on the present disclosure embodiment, block **7568** may include little privilege verification, no privilege verification, or may include all applicable action privilege verification discussed already in FIG. **61**.

In yet another embodiment, special terms processing of FIG. **61** can be delayed until FIG. **75**B processing (e.g. block **7566** continues to a new block **7567** which continues to block **7568**). It may be advantageous to have new block **7567** elaborate/evaluate special terms at the MS of FIG. **75**B processing in some embodiments. In a further embodiment, a syntax or qualifier can be used to differentiate where to perform special term elaboration/evaluation.

Thereafter, if block **7570** determines the action for execution is acceptable (and perhaps privileged, or privileged per source, or there was no check necessary), then block **7572** invokes the execute action procedure of FIG. **62** with the action (command, operand, and any parameter(s)), completes at block **7574** an acknowledgement to the originating MS of the data received at block **7556**, and block **7576** sends/broadcasts the acknowledgement (ack), before continuing back to block **7556** for the next incoming execution request data. Block **7576** 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 **7570** determines the data is not acceptable/privileged, then processing continues directly back to block **7556**. 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 **7564**, if it is determined that the execution data is for processing a particular atomic command, then processing continues to block **7578**. Block **7578** accesses the command (e.g. send), the operand, and parameters from the data received. Thereafter, block **7580** accesses privileges for each of the parts (command, operand, parameters) to ensure the source has proper privileges for running the atomic command at the MS of FIG. **75**B processing. Depending on embodiments, block **7580** may include evaluating the command for elaborating special terms and/or expressions as described for FIG. **61** (blocks **6140** through **6154**), although the preferred embodiment preferably already did that prior to transmitting the command for execution. However, in some embodiments where privileges are only maintained locally, the privilege processing of FIG. **61** would be required at block **7580** to check privileges where appropriate in processing the command. In such embodiments, FIG. **61** would process local actions as disclosed, but would not process actions known to be for remote execution (i.e. Host specification) since a FIG. **75**B embodiment would include FIG. **61** processing for performing privilege check processing to determine that sufficient privileges are granted. Thus, depending on the present disclosure embodiment, block **7580** may include little privilege verification, no privilege verification, or may include all applicable action privilege verification discussed already in FIG. **61**.

In yet another embodiment, special terms processing of FIG. **61** can be delayed until FIG. **75**B processing (e.g. block **7566** continues to a new block **7567** which continues to block **7568**). It may be advantageous to have new block **7567** elaborate/evaluate special terms at the MS of FIG. **75**B processing in some embodiments. In a further embodiment, a syntax or qualifier can be used to differentiate where to perform special term elaboration/evaluation.

Thereafter, if block **7582** determines the command (Command, Operand, Parameters) for execution is acceptable (and perhaps privileged, or privileged per source, or there was no check necessary), then block **7584** performs the command locally at the MS of FIG. **75**A processing. Thereafter, block **7586** checks if a response is needed as a result of command (e.g. Find command) processing at block **7584**. If block **7586** determines a response is to be sent back to the originating MS, **7574** completes a response to the originating MS of the data received at block **7556**, and block **7576** sends/broadcasts the response, before continuing back to block **7556** for the next incoming execution request data. Block **7576** sends/broadcasts the response containing appropriate command results (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 a response with a send.

If block **7586** determines a response is not to be sent back to the originating MS, then processing continues directly back to block **7556**. If block **7582** determines the data is not acceptable/privileged, then processing continues back to block **7556**. For security reasons, it is best not to respond with an error. It is best to ignore inappropriate (e.g. unprivileged, unwarranted) data entirely. In another embodiment, an error may be returned to the sender for appropriate error processing and reporting.

Blocks **7578** through **7584** are presented generically so that specific atomic command descriptions below provide appropriate interpretation and processing. The actual implementation may replace blocks **7578** through **7584** with programming case statement conditional execution for each atomic command supported.

Referring back to block **7562**, if it is determined that the data is not valid for the MS of FIG. **75** processing, processing continues back to block **7556**. Referring back to block **7558**, if a worker thread termination request was found at queue **26**, then block **7586** decrements the RxED worker thread count by 1 (using appropriate semaphore access (e.g. RxED-Sem)), and RxED thread processing terminates at block **7588**. Block **7586** may also check the RxED-Ct value, and signal the RxED process parent thread that all worker threads are terminated when RxED-Ct equals zero (0).

Block **7576** causes sending/broadcasting data **1302** containing CK **1304**, depending on the type of MS, wherein CK **1304** contains ack/response 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 **7576** processing, will place ack/response 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. **75**B sends/broadcasts new ack/response data **1302**.

In an alternate embodiment, remote action and/or atomic command 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. **75**B processing. This would enable calculating a TDOA measurement while receiving data (e.g. actions or atomic command) 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, just as discussed with FIG. **44**B above (e.g. mail application, callback function API, etc). Thus, there are well known methods for processing data in context of this disclosure for receiving remote actions and/or atomic command data from an originating MS to a receiving MS, for example when using email. Similarly, as described above, SMS messages can be used to communicate data, albeit at smaller data exchange sizes. The sending MS may break up larger portions

of data which can be sent as parse-able text 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). FIGS. **75**A and **75**B are an embodiment of MS to MS communications, referred to with the acronym MS2MS.

FIG. **62** depicts a flowchart for describing a preferred embodiment of a procedure for performing an action corresponding to a configured command, namely an ExecuteAction procedure. Only a small number of commands are illustrated. The procedure starts at block **6202** and continues to block **6204** where parameters of the Command, Operand, and Parameters are accessed (see BNF grammar), depending on an embodiment (e.g. parameters passed by reference or by value). Preferably, FIG. **62** procedure processing is passed parameters by reference (i.e. by address) so they are accessed as needed by FIG. **62** processing. Block **6204** continues to block **6206**.

If it is determined at block **6206** that the action atomic command is a send command, then processing continues to block **6208** where the send command action procedure of FIG. **63**A is invoked. The send command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the send command action procedure, block **6208** continues to block **6256**. Block **6256** returns to the calling block of processing (e.g. block **6158**) that invoked FIG. **62** processing. If block **6206** determines the action atomic command is not a send command, then processing continues to block **6210**. If it is determined at block **6210** that the action atomic command is a notify command, then processing continues to block **6212** where the notify command action procedure of FIG. **64**A is invoked. The notify command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the notify command action procedure, block **6212** continues to block **6256**. If block **6210** determines the action atomic command is not a notify command, then processing continues to block **6214**. If it is determined at block **6214** that the action atomic command is a compose command, then processing continues to block **6216** where the compose command action procedure of FIG. **65**A is invoked. The compose command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the compose command action procedure, block **6216** continues to block **6256**. If block **6214** determines the action atomic command is not a compose command, then processing continues to block **6218**. If it is determined at block **6218** that the action atomic command is a connect command, then processing continues to block **6220** where the connect command action procedure of FIG. **66**A is invoked. The connect command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the connect command action procedure, block **6220** continues to block **6256**. If block **6218** determines the action atomic command is not a connect command, then processing continues to block **6222**. If it is determined at block **6222** that the action atomic command is a find command, then processing continues to block **6224** where the find command action procedure of FIG. **67**A is invoked. The find command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the find command action procedure, block **6224** continues to block **6256**. If block **6222** determines the action atomic command is not a find command, then processing continues to block **6226**. If it is determined at block **6226** that the action atomic command is an invoke command, then processing continues to block **6228** where the invoke command action procedure of FIG. **68**A is invoked. The invoke command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the invoke command action procedure, block **6228** continues to block **6256**. If block **6226** determines the action atomic command is not an invoke command, then processing continues to block **6230**. If it is determined at block **6230** that the action atomic command is a copy command, then processing continues to block **6232** where the copy command action procedure of FIG. **69**A is invoked. The copy command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the copy command action procedure, block **6232** continues to block **6256**. If block **6230** determines the action atomic command is not a copy command, then processing continues to block **6234**. If it is determined at block **6234** that the action atomic command is a discard command, then processing continues to block **6236** where the discard command action procedure of FIG. **70**A is invoked. The discard command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the discard command action procedure, block **6236** continues to block **6256**. If block **6234** determines the action atomic command is not a discard command, then processing continues to block **6238**. If it is determined at block **6238** that the action atomic command is a move command, then processing continues to block **6240** where the move command action procedure of FIG. **71**A is invoked. The move command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the move command action procedure, block **6240** continues to block **6256**. If block **6238** determines the action atomic command is not a move command, then processing continues to block **6242**. If it is determined at block **6242** that the action atomic command is a store command, then processing continues to block **6244** where the store command action procedure of FIG. **72**A is invoked. The store command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the store command action procedure, block **6244** continues to block **6256**. If block **6242** determines the action atomic command is not a store command, then processing continues to block **6246**. If it is determined at block **6246** that the action atomic command is an administrate command, then processing continues to block **6248** where the administrate command action procedure of FIG. **73**A is invoked. The administrate command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the administrate command action procedure, block **6248** continues to block **6256**. If block **6246** determines the action atomic command is not an administrate command, then processing continues to block **6250**. If it is determined at block **6250** that the action atomic command is a change command, then processing continues to block **6252** where the

change command action procedure of FIG. **74**A is invoked. The change command action procedure is invoked with parameters including the passed parameters of Operand and Parameters discussed for block **6204**. Upon return from the change command action procedure, block **6252** continues to block **6256**. If block **6250** determines the action atomic command is not a change command, then processing continues to block **6254** for handling other supported action atomic commands on the MS. There are many commands that can be implemented on a MS. Block **6254** continues to block **6256** for processing as already described. FIGS. **60** through **62** describe action processing for recognized events to process WDRs.

FIGS. **63**A through **74**C document a MS toolbox of useful actions. FIGS. **63**A through **74**C are in no way intended to limit LBX functionality with a limited set of actions, but rather to demonstrate a starting list of tools. New atomic commands and operands can be implemented with contextual "plug-in" processing code, API plug-in processing code, command line invoked plug-in processing code, local data processing system (e.g. MS) processing code, MS2MS plug-in processing code, or other processing, all of which are described below. The "know how" of atomic commands is preferably isolated for a variety of "plug-in" processing. The charter and privilege platform is designed for isolating the complexities of privileged actions to "plug-in" methods of new code (e.g. for commands and/or operands) wherever possible.

Together with processing disclosed above, provided is a user friendly development platform for quickly building LBX applications wherein the platform enables conveniently enabled LBX application interoperability and processing, including synchronized processing, across a plurality of MSs. Some commands involve a plurality of MSs and/or data processing systems. Others don't explicitly support a plurality of MSs and data processing systems, however that is easily accomplished for every command since a single charter expression can cause a plurality of actions anyway. For example, if a command does not support a plurality of MSs in a single command action, the plurality of MSs is supported with that command through specifying a plurality of identical command actions in the charter configuration for each desired MS. Actions provided in this LBX release enable a rich set of LBX features and functionality for:

- Desired local MS LBX processing;
- Desired peer MS LBX processing relative permissions provided; and
- Desired MS LBX processing from a global perspective of a plurality of MSs. MS operating system resources of memory, storage, semaphores, and applications and application data is made accessible to other MSs as governed by permissions. Thus, a single MS can become a synchronization point for any plurality of MSs, and synchronized processing can be achieved across a plurality of independently operating MSs.

There are many different types of actions, commands, operands, parameters, etc that are envisioned, but embodiments share at least the following fundamental characteristics:

1) Syntax is governed by the LBX BNF grammar;
2) Command is a verb for performing an action (i.e. atomic command);
3) Operand is an object which provides what is acted upon by the Command—e.g. brings context of how to process Command (i.e. atomic operand); and
4) Parameters are anticipated by a combination of Command and Operand. Each parameter can be a constant, of any data type, or a resulting evaluation of any arithmetic or semantic expression, which may include atomic terms, WDRTerms, AppTerms, atomic operators, etc (see BNF grammar). Parameter order, syntax, semantics, and variances of specification(s) are anticipated by processing code. Obvious error handling is incorporated in action processing.

Syntax and reasonable validation should be performed at the time of configuration, although it is preferable to check for errors at run time of actions as well. Various embodiments may or may not validate at configuration time, and may or may not validate at action processing time. Validation should be performed at least once to prevent run time errors from occurring. Obvious error handling is assumed present when processing commands, such error handling preferably including the logging of the error to LBX History **30** and/or notifying the user of the error with, or without, request for the user to acknowledge the reporting of error.

FIGS. **63**A through **74**C are organized for presenting three (3) parts to describing atomic commands (e.g. **63**A, **63**B (e.g. **63**B-**1** through **63**B-**7**), **63**C):

- #A=describes preferred embodiment of command action processing;
- #B=describes LBX command processing for some operands; and
- #C=describes one embodiment of command action processing.

Some of the #A figures highlight diversity for showing different methods of command processing while highlighting that some of the methods are interchangeable for commands (e.g. Copy and Discard processing). Also the terminology "application" and "executable" are used interchangeably to represent an entity of processing which can be started, terminated, and have processing results. Applications (i.e. executables) can be started as a contextual launch, custom launch through an API or command line, or other launch method of an executable for processing.

Atomic command descriptions are to be interpreted in the broadest sense, and some guidelines when reading the descriptions include:

1) Any action (Command, Operand, Parameters) can include an additional parameter, or use an existing parameter if appropriate (e.g. attributes) to warn an affected user that the action is pending (i.e. about to occur). The warning provides the user with informative information about the action and then waits for the user to optionally accept (confirm) the action for processing, or cancel it;
2) In alternate embodiments, an email or similar messaging layer may be used as a transport for conveying and processing actions between systems. As disclosed above, characteristic(s) of the transported distribution will distinguish it from other distributions for processing uniquely at the receiving system(s);
3) Identities (e.g. sender, recipient, source, system, etc) which are targeted data processing systems for processing are described as MSs, but can be a data processing system other than a MS in some contexts provided the identified system has processing as disclosed;
4) Obvious error handling is assumed and avoided in the descriptions.

The reader should cross reference/compare operand descriptions in the #B matrices for each command to appreciate full exploitation of the Operand, options, and intended embodiments since descriptions assume information found in other commands is relevant across commands. Some operand description information may have been omitted from a com-

mand matrix to prevent obvious duplication of information already described for the same operand in another command.

FIG. **63**A depicts a flowchart for describing a preferred embodiment of a procedure for Send command action processing. There are three (3) primary methodologies for carrying out send command processing:

1) Using email or similar messaging layer as a transport layer;
2) Using a MS to MS communications (MS2MS) of FIGS. **75**A and **75**B; or
3) Processing the send command locally.

In various embodiments, any of the send command Operands can be implemented with either one of the methodologies, although there may be a preference of which methodology is used for which Operand. Atomic send command processing begins at block **6302**, continues to block **6304** for accessing parameters of send command "Operand" (BNF Grammar Operand) and "Parameters" (BNF Grammar Parameters), and then to block **6306** for checking which "Operand" was passed. If block **6306** determines the "Operand" indicates to use email as the mechanism for performing the send command, then block **6308** checks if a sender parameter was specified. If block **6308** determines a sender was specified, processing continues to block **6312**, otherwise block **6310** defaults one (e.g. valid email address for this MS) and then processing continues to block **6312**. Block **6312** checks if a subject parameter was specified. If block **6312** determines a subject was specified, processing continues to block **6316**, otherwise block **6314** defaults one (e.g. subject line may be used to indicate to email receive processing that this is a special email for performing atomic command (e.g. send command) processing), and then processing continues to block **6316**. Block **6314** may specify a null email subject line. Block **6316** checks if an attributes parameter was specified. If block **6316** determines attributes were specified, processing continues to block **6320**, otherwise block **6318** defaults attributes (e.g. confirmation of delivery, high priority, any email Document Interchange Architecture (DIA) attributes or profile specifications, etc) and then processing continues to block **6320**. Block **6318** may use email attributes to indicate that this is a special email for send command processing while using the underlying email transport to handle the delivery of information. Block **6320** checks if at least one recipient parameter was specified. If block **6320** determines at least one recipient was specified, processing continues to block **6324**, otherwise block **6322** defaults one (e.g. valid email address for this MS) and then processing continues to block **6324**. Block **6322** may specify a null recipient list so as to cause an error in later processing (detected at block **6324**).

Block **6324** validates "Parameters", some of which may have been defaulted in previous blocks (**6310**, **6314**, **6318** and **6322**), and continues to block **6326**. If bock **6326** determines there is an error in "Parameters", then block **6328** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller (invoker) at block **6334**. If block **6326** determines that "Parameters" are in good order for using the email transport, then block **6330** updates an email object in context for the send command "Operand" and "Parameters", block **6332** uses a send email interface to send the email, and block **6334** returns to the caller (e.g. block **6208**). Block **6330** can use the attributes parameter to affect how "Parameters" is to be interpreted. The attributes parameter may be modified, and can be used by any processes which receive the sent distribution. Those skilled in the art know well known email send interfaces (e.g. APIs) depending on a software development environment. The email interface used at block **6332** will be one suitable for the underlying operating system and available development environments, for example, a standardized SMTP interface. In a C# environment, an SMTP email interface example is:

```
. . .
SmtpClient smtpCl=new SmtpClient(SMTP_SERVER_NAME);
. . .
smtpCl.UseDefaultCredentials=true;
. . .
MailMessage objMsg;
. . .
objMsg=new MailMessage(fromAddr, toAddr, subjLn, emailBod);
. . .
smtpCl.Send(objMsg);
objMsg.Dispose( );
. . .
```

Those skilled in the art recognize other interfaces of similar messaging capability for carrying out the transport of an action (e.g. Send command). Email is a preferred embodiment. While there are Send command embodiments that make using an existing transport layer (e.g. email) more suitable than not, even the most customized Send command Operands can use email (instead of MS2MS) by implementing one or more recognizable signature(s), indication(s), or the like, of/in the email distribution to be used for informing a receiving email system to treat the email uniquely for carrying out the present disclosure. Depending on the embodiment, integrated processing code is maintained/built as part of the email system, or processing code is "plugged" ("hooked") into an existing email system in an isolated third party manner. Regardless, the email system receiving the present disclosure email will identify the email as being one for special processing. Then, email contents is parsed out and processed according to what has been requested.

In embodiments where Send command Operands are more attractively implemented using an existing transport layer (e.g. email), those send commands can also be sent with MS2MS encoded in data packet(s) that are appropriate for processing.

Referring back to block **6306**, if it is determined that the "Operand" indicates to not use an email transport (e.g. use a MS2MS transport for performing the send command, or send command is to be processed locally), then block **6336** checks if a sender parameter was specified. If block **6336** determines a sender was specified, processing continues to block **6340**, otherwise block **6338** defaults one (e.g. valid MS ID) and then processing continues to block **6340**. Block **6340** checks if a subject message parameter was specified. If block **6340** determines a subject message was specified, processing continues to block **6344**, otherwise block **6342** defaults one, and then processing continues to block **6344**. Block **6342** may specify a null message. Block **6344** checks if an attributes parameter was specified. If block **6344** determines attributes were specified, processing continues to block **6348**, otherwise block **6346** defaults attributes (e.g. confirmation of delivery, high priority, etc) and then processing continues to block **6348**. Block **6348** checks if at least one recipient parameter was specified. If block **6348** determines at least one recipient was specified, processing continues to block **6352**, otherwise block **6350** defaults one (e.g. valid ID for this MS) and then processing continues to block **6352**. Block **6350** may specify a null recipient list so as to cause an error in later processing (detected at block **6352**).

Block **6352** validates "Parameters", some of which may have been defaulted in previous blocks (**6338**, **6342**, **6346** and **6350**), and continues to block **6354**. If bock **6354** determines

there is an error in "Parameters", then block **6356** handles the error appropriately (e.g. log error to LBX History and/or notify user) and processing returns to the caller (invoker) at block **6334**. If block **6354** determines that "Parameters" are in good order, then block **6358** updates a data object in context for the send command "Operand" and "Parameters", and block **6360** begins a loop for delivering the data object to each recipient. Block **6360** gets the next (or first) recipient from the recipient list and processing continues to block **6362**.

If block **6362** determines that all recipients have been processed, then processing returns to the caller at block **6334**, otherwise block **6364** checks the recipient to see if it matches the ID of the MS of FIG. **63**A processing (i.e. this MS). If block **6364** determines the recipient matches this MS, then block **6366** (see FIG. **63**B discussions) performs the atomic send command locally and processing continues back to block **6360** for the next recipient. If block **6364** determines the recipient is an other MS, block **6368** prepares parameters for FIG. **75**A processing, and block **6370** invokes the procedure of FIG. **75**A for sending the data (send command, operand and parameters) to the other MS. Processing then continues back to block **6360** for the next recipient. Blocks **6366**, **6368**, and **7584** can use the attributes parameter to affect how "Parameters" is to be interpreted. The attributes parameter may be modified, and can be used by any processes which receive the send result.

MS2MS processing is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs sending data for the send command to a remote MS, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for the send command. Block **7584** processes the send command locally (like block **6366**—see FIG. **63**B).

In FIG. **63**A, "Parameters" for the atomic send command in accordance with the "Operand" were shown to be validated for being properly privileged prior to FIG. **63**A processing (by FIG. **61** processing). However, an alternate embodiment could move some or all applicable privilege validation to FIG. **63**A in context of where the "Parameters" are processed. Also, some embodiments may not validate "Parameters" since they (or some reasonable subset thereof) can be understood to be in good order by the time FIG. **63**A processing occurs (e.g. no blocks **6308** through **6328** and/or **6336** through **6356** required). In yet another embodiment, no defaulting or some defaulting of parameters is implemented. In some embodiments, any subset of send commands will utilize email distributions for processing between MSs. In other embodiments, any subset of send commands will utilize FIGS. **75**A and **75**B for processing between MSs. Operations of the send command can be carried out regardless of the transport that is actually used to perform the send command.

FIGS. **63**B-**1** through **63**B-**7** depicts a matrix describing how to process some varieties of the Send command (e.g. as processed at blocks **6366** and **7584**). Each row in the matrix describes processing apparatus and/or methods for carrying out command processing for certain operands (see FIG. **34**D for the Operand which matches the number in the first column). The second column shows the Preferred Methodology (PM) for carrying out Send command processing:

E=Email transport preferably used (blocks **6308** through **6332**);

O=Other processing (MS2MS or local) used (blocks **6336** through **6370**).

Any of the Send command operand combinations can be carried out with either of the methodologies. The second column shows a preferred methodology (PM). The third column describes processing which is placed into flowchart embodiments. There are many embodiments derived from the Send processing descriptions without departing from the spirit and scope of the disclosure. Descriptions are self explanatory.

With reference back to FIGS. **31**A through **31**E, note that the column of information headed by "**101**" represents the parameters applicable for the Send command. The Send command has the following parameters, all of which are interpreted in context of the Operand:

first parameter(s)=These are required, and are in context of the Operand;

sender=The sender of the Send command, typically tied to the originating identity of the action (e.g. email address or MS ID). A different sender can be specified if there is an applicable privilege in place, or if impersonation has been granted;

msg/subj=A message or subject associated with Send command;

attributes=Indicators for more detailed interpretation of Send command parameters and/or indicators for attributes to be interpreted by external (e.g. receiving) processes affected by the Send command result (e.g. handled appropriately by block **7584** or receiving email system);

recipient(s)=One or more destination identities for the Send command (e.g. email address or MS ID).

FIG. **63**C depicts a flowchart for describing one embodiment of a procedure for Send command action processing, as derived from the processing of FIG. **63**A. All operands are implemented, and each of blocks S**04** through S**54** can be implemented with any one of the methodologies described with FIG. **63**A, or any one of a blend of methodologies implemented by FIG. **63**C.

FIG. **64**A depicts a flowchart for describing a preferred embodiment of a procedure for Notify command action processing. The Alert command and Notify command provide identical processing. There are three (3) primary methodologies for carrying out notify command processing:

1) Using email or similar messaging layer as a transport layer;

2) Using a MS to MS communications (MS2MS) of FIGS. **75**A and **75**B; or

3) Processing the notify command locally.

In various embodiments, any of the notify command Operands can be implemented with either one of the methodologies, although there may be a preference of which methodology is used for which Operand. Atomic notify command processing begins at block **6402**, continues to block **6404** for accessing parameters of notify command "Operand" (BNF Grammar Operand) and "Parameters" (BNF Grammar Parameters), and then to block **6406** for checking which "Operand" was passed. If block **6406** determines the "Operand" indicates to use email as the mechanism for performing the notify command, then block **6408** checks if a sender parameter was specified. If block **6408** determines a sender was specified, processing continues to block **6412**, otherwise block **6410** defaults one (e.g. valid email address for this MS) and then processing continues to block **6412**. Block **6412** checks if a subject parameter was specified. If block **6412** determines a subject was specified, processing continues to block **6416**, otherwise block **6414** defaults one (e.g. subject line may be used to indicate to email receive processing that this is a special email for performing atomic command (e.g. notify command) processing), and then processing continues to block **6416**. Block **6414** may specify a null email subject line. Block **6416** checks if an attributes parameter was specified. If block **6416** determines attributes were specified, processing continues to block **6420**, otherwise block **6418** defaults attributes (e.g. confirmation of delivery, high prior-

ity, any email DIA attributes or profile specifications, etc) and then processing continues to block **6420**. Block **6418** may use email attributes to indicate that this is a special email for notify command processing while using the underlying email transport to handle the delivery of information. Block **6420** checks if at least one recipient parameter was specified. If block **6420** determines at least one recipient was specified, processing continues to block **6424**, otherwise block **6422** defaults one (e.g. valid email address for this MS) and then processing continues to block **6424**. Block **6422** may specify a null recipient list so as to cause an error in later processing (detected at block **6424**).

Block **6424** validates "Parameters", some of which may have been defaulted in previous blocks (**6410**, **6414**, **6418** and **6422**), and continues to block **6426**. If bock **6426** determines there is an error in "Parameters", then block **6428** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller (invoker) at block **6434**. If block **6426** determines that "Parameters" are in good order for using the email transport, then block **6430** updates an email object in context for the notify command "Operand" and "Parameters", block **6432** uses a send email interface to notify through email, and block **6434** returns to the caller (e.g. block **6212**). Block **6430** can use the attributes parameter to affect how "Parameters" is to be interpreted. The attributes parameter may be modified, and can be used by any processes which receive the notify. The email interface used at block **6432** will be one suitable for the underlying operating system and available development environments, for example, a standardized SMTP interface, and other messaging capability, as described above for FIG. **63**A.

While there are Notify command embodiments that make using an existing transport layer (e.g. email) more suitable than not, even the most customized Notify command Operands can use email (instead of MS2MS) by implementing one or more recognizable signature(s), indication(s), or the like, of/in the email distribution to be used for informing a receiving email system to treat the email uniquely for carrying out the present disclosure. Depending on the embodiment, integrated processing code is maintained/built as part of the email system, or processing code is "plugged" ("hooked") into an existing email system in an isolated third party manner. Regardless, the email system receiving the present disclosure email will identify the email as being one for special processing. Then, email contents is parsed out and processed according to what has been requested.

In embodiments where Notify command Operands are more attractively implemented using an existing transport layer (e.g. email), those notify commands can also be sent with MS2MS encoded in data packet(s) that are appropriate for processing.

Referring back to block **6406**, if it is determined that the "Operand" indicates to not use an email transport (e.g. use a MS2MS transport for performing the notify command, or notify command is to be processed locally), then block **6436** checks if a sender parameter was specified. If block **6436** determines a sender was specified, processing continues to block **6440**, otherwise block **6438** defaults one (e.g. valid MS ID) and then processing continues to block **6440**. Block **6440** checks if a subject message parameter was specified. If block **6440** determines a subject message was specified, processing continues to block **6444**, otherwise block **6442** defaults one, and then processing continues to block **6444**. Block **6442** may specify a null message. Block **6444** checks if an attributes parameter was specified. If block **6444** determines attributes were specified, processing continues to block **6448**, otherwise block **6446** defaults attributes (e.g. confirmation of delivery, high priority, etc) and then processing continues to block **6448**. Block **6448** checks if at least one recipient parameter was specified. If block **6448** determines at least one recipient was specified, processing continues to block **6452**, otherwise block **6450** defaults one (e.g. valid ID for this MS) and then processing continues to block **6452**. Block **6450** may specify a null recipient list so as to cause an error in later processing (detected at block **6452**).

Block **6452** validates "Parameters", some of which may have been defaulted in previous blocks (**6438**, **6442**, **6446** and **6450**), and continues to block **6454**. If bock **6454** determines there is an error in "Parameters", then block **6456** handles the error appropriately (e.g. log error to LBX History and/or notify user) and processing returns to the caller (invoker) at block **6434**. If block **6454** determines that "Parameters" are in good order, then block **6458** updates a data object in context for the notify command "Operand" and "Parameters", and block **6460** begins a loop for delivering the data object to each recipient. Block **6460** gets the next (or first) recipient from the recipient list and processing continues to block **6462**.

If block **6462** determines that all recipients have been processed, then processing returns to the caller at block **6434**, otherwise block **6464** checks the recipient to see if it matches the ID of the MS of FIG. **64**A processing (i.e. this MS). If block **6464** determines the recipient matches this MS, then block **6466** (see FIG. **64**B discussions) performs the atomic notify command locally and processing continues back to block **6460** for the next recipient. If block **6464** determines the recipient is an other MS, block **6468** prepares parameters for FIG. **75**A processing, and block **6470** invokes the procedure of FIG. **75**A for sending the data (notify command, operand and parameters) to the other MS. Processing then continues back to block **6460** for the next recipient. Blocks **6466**, **6468**, and **7584** can use the attributes parameter to affect how "Parameters" is to be interpreted. The attributes parameter may be modified, and can be used by any processes which receive the notify result.

MS2MS processing is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs sending data for the notify command to a remote MS, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for the notify command. Block **7584** processes the notify command locally (like block **6466**—see FIG. **64**B).

In FIG. **64**A, "Parameters" for the atomic notify command in accordance with the "Operand" were shown to be validated for being properly privileged prior to FIG. **64**A processing (by FIG. **61** processing). However, an alternate embodiment could move some or all applicable privilege validation to FIG. **64**A in context of where the "Parameters" are processed. Also, some embodiments may not validate "Parameters" since they (or some reasonable subset thereof can be understood to be in good order by the time FIG. **64**A processing occurs (e.g. no blocks **6408** through **6428** and/or **6436** through **6456** required). In yet another embodiment, no defaulting or some defaulting of parameters is implemented. In some embodiments, any subset of notify commands will utilize email distributions for processing between MSs. In other embodiments, any subset of notify commands will utilize FIGS. **75**A and **75**B for processing between MSs. Operations of the notify command can be carried out regardless of the transport that is actually used to perform the notify command.

FIGS. **64**B-**1** through **64**B-**4** depicts a matrix describing how to process some varieties of the Notify command (e.g. as processed at blocks **6466** and **7584**). Each row in the matrix describes processing apparatus and/or methods for carrying out command processing for certain operands (see FIG. **34**D

for the Operand which matches the number in the first column). The second column shows the Preferred Methodology (PM) for carrying out Notify command processing:

E=Email transport preferably used (blocks **6408** through **6432**);

O=Other processing (MS2MS or local) used (blocks **6436** through **6470**).

Any of the Notify command operand combinations can be carried out with either of the methodologies. The second column shows a preferred methodology (PM). The third column describes processing which is placed into flowchart embodiments. There are many embodiments derived from the Notify processing descriptions without departing from the spirit and scope of the disclosure. Descriptions are self explanatory.

With reference back to FIGS. **31**A through **31**E, note that the column of information headed by "**103**" represents the parameters applicable for the Notify command. The Notify command has the following parameters, all of which are interpreted in context of the Operand:

first parameter(s)=These are required, and are in context of the Operand;

sender=The sender of the Notify command, typically tied to the originating identity of the action (e.g. email address or MS ID). A different sender can be specified if there is an applicable privilege in place, or if impersonation has been granted;

msg/subj=A message or subject associated with Notify command;

attributes=Indicators for more detailed interpretation of Notify command parameters and/or indicators for attributes to be interpreted by external (e.g. receiving) processes affected by the Notify command result (e.g. handled appropriately by block **7584** or receiving email system);

recipient(s)=One or more destination identities for the Notify command (e.g. email address or MS ID).

FIG. **64**C depicts a flowchart for describing one embodiment of a procedure for Notify command action processing, as derived from the processing of FIG. **64**A. All operands are implemented, and each of blocks N**04** through N**54** can be implemented with any one of the methodologies described with FIG. **64**A, or any one of a blend of methodologies implemented by FIG. **64**C.

FIG. **65**A depicts a flowchart for describing a preferred embodiment of a procedure for Compose command action processing. The Make command and Compose command provide identical processing. There are three (3) primary methodologies for carrying out compose command processing:

1) Launching an application, executable, or program with a standard contextual object type interface;
2) Custom launching of an application, executable, or program; or
3) Processing the compose command through a MS operating system interface.

In various embodiments, any of the compose command Operands can be implemented with either one of the methodologies, although there may be a preference of which methodology is used for which Operand. Atomic compose command processing begins at block **6502**, continues to block **6504** for accessing parameters of compose command "Operand" (BNF Grammar Operand) and "Parameters" (BNF Grammar Parameters), and then to block **6506** for checking which "Operand" was passed. If block **6506** determines the "Operand" indicates to launch with a standard contextual object type interface, then parameter(s) are validated at block **6508** and block **6510** checks the result. If block **6510** determines there was at least one error, then block **6512** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller (invoker) at block **6514**. If block **6510** determines there were no parameter errors, then block **6516** interfaces to the MS operating system for the particular object passed as a parameter. Block **6516** may prepare parameters in preparation for the Operating System (O/S) contextual launch, for example if parameters are passed to the application which is invoked for composing the object. Processing leaves block **6516** and returns to the caller (invoker) at block **6514**.

An example of block **6516** is similar to the Microsoft Windows XP (Microsoft and Windows XP are trademarks of Microsoft corp.) O/S association of applications to file types for convenient application launch. For example, a user can double click a file (e.g. when viewing file system) from Window Explorer and the appropriate application will be launched for opening the file, assuming an application has been properly registered for the file type of the file opened. In a Windows graphical user interface scenario, registration of an application to the file type is achieved, for example, from the user interface with the "File Types" tab of the "Folder Options" option of the "File Types" pulldown of the Windows Explorer interface. There, a user can define file types and the applications which are to be launched when selecting/invoking (e.g. double clicking) the file type from the file system. Alternatively, an O/S API or interface may be used to configure an object to associate to a launch-able executable for handling the object. In this same scheme, the MS will have a similar mechanism whereby an association of an application to a type of object (e.g. file type) has been assigned. Block **6516** makes use of the system interface for association which was set up outside of present disclosure processing (e.g. via MS O/S).

Referring back to block **6506**, if it is determined the "Operand" does not indicate to launch with a standard contextual object type interface, processing continues to block **6518**. If block **6518** determines the "Operand" indicates to perform a custom launch, then parameter(s) are validated at block **6520** and block **6522** checks the result. If block **6522** determines there was at least one error, then block **6524** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller (invoker) at block **6514**. If block **6522** determines there were no parameter errors, then processing continues to block **6526**.

If block **6526** determines the custom launch is not to use an Application Programming Interface (API) to launch the applicable application for composing the object passed as a parameter, then block **6528** prepares a command string for launching the particular application, block **6530** invokes the command string for launching the application, and processing continues to block **6514** for returning to the caller.

If block **6526** determines the custom launch is to use an Application Programming Interface (API) to launch the applicable application for composing the object passed as a parameter, then block **6532** prepares any API parameters as necessary, block **6534** invokes the API for launching the application, and processing continues to block **6514** for returning to the caller.

Referring back to block **6518**, if it is determined that the "Operand" indicates to perform the compose command locally (e.g. use operating system interface (e.g. set semaphore, program object, data, signal, etc)), then parameter(s) are validated at block **6536** and block **6538** checks the result. If block **6538** determines there was at least one error, then block **6540** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to

the caller (invoker) at block **6514**. If block **6538** determines there were no parameter errors, then block **6542** performs the compose command, and block **6514** returns to the caller.

In FIG. **65**A, "Parameters" for the atomic compose command in accordance with the "Operand" were shown to be validated for being properly privileged prior to FIG. **65**A processing (by FIG. **61** processing). However, an alternate embodiment could move some or all applicable privilege validation to FIG. **65**A in context of where the "Parameters" are processed. Also, some embodiments may not validate "Parameters" since they (or some reasonable subset thereof) can be understood to be in good order by the time FIG. **65**A processing occurs (e.g. no blocks **6510**/**6512** and/or **6522**/**6524** and/or **6538**/**6540** required). In yet another embodiment, some defaulting of parameters is implemented.

FIGS. **65**B-**1** through **65**B-**7** depicts a matrix describing how to process some varieties of the Compose command (e.g. as resulting after blocks **6516**, **6534** and **6542**). Each row in the matrix describes processing apparatus and/or methods for carrying out command processing for certain operands (see FIG. **34**D for the Operand which matches the number in the first column). The second column shows the Preferred Methodology (PM) for carrying out Compose command processing:

S=Standard contextual launch used (blocks **6508** through **6516**);

C=Custom launch used (blocks **6520** through **6534**);

O=Other processing (O/S interface) used (blocks **6536** through **6542**).

Any of the Compose command operand combinations can be carried out with either of the methodologies. The second column shows a preferred methodology (PM). The third column describes processing which is placed into flowchart embodiments. There are many embodiments derived from the Compose processing descriptions without departing from the spirit and scope of the disclosure. Descriptions are self explanatory.

With reference back to FIGS. **31**A through **31**E, note that the column of information headed by "105" represents the parameters applicable for the Compose command. The Compose command has the following parameters, all of which are interpreted in context of the Operand:

first parameter(s)=These are required, and are in context of the Operand;

sender=The sender of the Compose command, typically tied to the originating identity of the action (e.g. email address or MS ID). A different sender can be specified if there is an applicable privilege in place, or if impersonation has been granted;

msg/subj=A message or subject associated with Compose command;

attributes=Indicators for more detailed interpretation of Compose command parameters and/or indicators for attributes to be interpreted by external (e.g. receiving) processes affected by the Compose command result;

recipient(s)=One or more destination identities for the Compose command (e.g. email address or MS ID).

Compose command data is preferably maintained to LBX history, a historical call log (e.g. outgoing when call placed), or other useful storage for subsequent use (some embodiments may include this processing where appropriate (e.g. as part of blocks **6516**, **6542**, etc)).

FIG. **65**C depicts a flowchart for describing one embodiment of a procedure for Compose command action processing, as derived from the processing of FIG. **65**A. All operands are implemented, and each of blocks P**04** through P**54** can be implemented with any one of the methodologies described with FIG. **65**A, or any one of a blend of methodologies implemented by FIG. **65**C.

FIG. **66**A depicts a flowchart for describing a preferred embodiment of a procedure for Connect command action processing. The Call command and Connect command provide identical processing. There are four (4) primary methodologies for carrying out connect command processing:

1) Launching an application, executable, or program with a standard contextual object type interface;
2) Custom launching of an application, executable, or program;
3) Processing the connect command through a MS operating system interface; or
4) Using a MS to MS communications (MS2MS) of FIGS. **75**A and **75**B.

In various embodiments, any of the connect command Operands can be implemented with either one of the methodologies, although there may be a preference of which methodology is used for which Operand. Atomic connect command processing begins at block **6602**, continues to block **6604** for accessing parameters of connect command "Operand" (BNF Grammar Operand) and "Parameters" (BNF Grammar Parameters), and then to block **6606** for checking which "Operand" was passed. If block **6606** determines the "Operand" indicates to launch with a standard contextual object type interface, then parameter(s) are validated at block **6608** and block **6610** checks the result. If block **6610** determines there was at least one error, then block **6612** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller (invoker) at block **6614**. If block **6610** determines there were no parameter errors, then block **6616** interfaces to the MS operating system for the particular object passed as a parameter. Block **6616** may prepare parameters in preparation for the O/S contextual launch, for example if parameters are passed to the application which is invoked. Processing leaves block **6616** and returns to the caller (invoker) at block **6614**.

An example of block **6616** is similar to the Microsoft Windows XP O/S association of applications to file types for convenient application launch, and is the same as processing of block **6516** described above. Block **6616** makes use of the system interface for association which was set up outside of present disclosure processing (e.g. via MS O/S).

Referring back to block **6606**, if it is determined the "Operand" does not indicate to launch with a standard contextual object type interface, processing continues to block **6618**. If block **6618** determines the "Operand" indicates to perform a custom launch, then parameter(s) are validated at block **6620** and block **6622** checks the result. If block **6622** determines there was at least one error, then block **6624** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller (invoker) at block **6614**. If block **6622** determines there were no parameter errors, then processing continues to block **6626**.

If block **6626** determines the custom launch is not to use an Application Programming Interface (API) to launch the applicable application for the object passed as a parameter, then block **6628** prepares a command string for launching the particular application, block **6630** invokes the command string for launching the application, and processing continues to block **6614** for returning to the caller.

If block **6626** determines the custom launch is to use an Application Programming Interface (API) to launch the applicable application for the object passed as a parameter, then block **6632** prepares any API parameters as necessary,

block **6634** invokes the API for launching the application, and processing continues to block **6614** for returning to the caller.

Referring back to block **6618**, if it is determined that the "Operand" indicates to perform the connect command locally (e.g. use operating system interface (e.g. set semaphore, program object, data, signal, etc)), or to use MS2MS for processing, then parameter(s) are validated at block **6636** and block **6638** checks the result. If block **6638** determines there was at least one error, then block **6640** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller (invoker) at block **6614**. If block **6638** determines there were no parameter errors, then block **6642** checks the operand for which processing to perform. If block **6642** determines that MS2MS processing is needed to accomplish processing, then block **6644** prepares parameters for FIG. **75**A processing, and block **6646** invokes the procedure of FIG. **75**A for sending the data (connect command, operand and parameters) for connect processing at the MS to connect. Processing then continues to block **6614**. MS2MS processing is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs sending data for the connect command to the remote MS for processing, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for the connect command. Block **7584** processes the connect command for connecting the MSs in context of the Operand. Referring back to block **6642**, if it is determined that MS2MS is not to be used, then block **6648** performs the connect command, and block **6614** returns to the caller.

In FIG. **66**A, "Parameters" for the atomic connect command in accordance with the "Operand" were shown to be validated for being properly privileged prior to FIG. **66**A processing (by FIG. **61** processing). However, an alternate embodiment could move some or all applicable privilege validation to FIG. **66**A in context of where the "Parameters" are processed. Also, some embodiments may not validate "Parameters" since they (or some reasonable subset thereof) can be understood to be in good order by the time FIG. **66**A processing occurs (e.g. no blocks **6610**/**6612** and/or **6622**/**6624** and/or **6638**/**6640** required). In yet another embodiment, some defaulting of parameters is implemented.

In the case of automatically dialing a phone number at a MS, there are known APIs to accomplish this functionality, depending on the MS software development environment, by passing at least a phone number to the MS API programmatically at the MS (e.g. see C# phone application APIs, J2ME phone APIs, etc). In a J2ME embodiment, you can place a call by calling the MIDP 2.0 platformRequest method inside the MIDlet class (e.g. platformRequest("tel://mobileNumber") will request the placing call functionality from the applicable mobile platform).

FIGS. **66**B-**1** through **66**B-**2** depicts a matrix describing how to process some varieties of the Connect command (e.g. as processed at blocks **6648** and **7584**). Each row in the matrix describes processing apparatus and/or methods for carrying out command processing for certain operands (see FIG. **34**D for the Operand which matches the number in the first column). The second column shows the Preferred Methodology (PM) for carrying out Connect command processing:

S=Standard contextual launch used (blocks **6608** through **6616**);

C=Custom launch used (blocks **6620** through **6634**);

O=Other processing (MS2MS or local) used (blocks **6636** through **6648**).

Any of the Connect command operand combinations can be carried out with either of the methodologies. The second column shows a preferred methodology (PM). The third column describes processing which is placed into flowchart embodiments. There are many embodiments derived from the Connect processing descriptions without departing from the spirit and scope of the disclosure. Descriptions are self explanatory.

With reference back to FIGS. **31**A through **31**E, note that the column of information headed by "**119**" represents the parameters applicable for the Connect command. The Connect command has the following parameters, all of which are interpreted in context of the Operand:

first parameter(s)=These are required, and are in context of the Operand;

sender=The sender of the Connect command, typically tied to the originating identity of the action (e.g. email address or MS ID). A different sender can be specified if there is an applicable privilege in place, or if impersonation has been granted;

msg/subj=A message or subject associated with Connect command;

attributes=Indicators for more detailed interpretation of Connect command parameters and/or indicators for attributes to be interpreted by external (e.g. receiving) processes affected by the Connect command result;

recipient(s)=One or more destination identities for the Connect command (e.g. email address or MS ID).

Connect command data is preferably maintained to LBX history, a historical call log (e.g. outgoing when call placed), or other useful storage for subsequent use (some embodiments may include this processing where appropriate (e.g. as part of blocks **6616**, **6648**, **7584**, etc)).

FIG. **66**C depicts a flowchart for describing one embodiment of a procedure for Connect command action processing, as derived from the processing of FIG. **66**A. All operands are implemented, and each of blocks T**04** through T**54** can be implemented with any one of the methodologies described with FIG. **66**A, or any one of a blend of methodologies implemented by FIG. **66**C.

FIG. **67**A depicts a flowchart for describing a preferred embodiment of a procedure for Find command action processing. The Search command and Find command provide identical processing. There are four (4) primary methodologies for carrying out find command processing:

1) Launching an application, executable, or program with a standard contextual object type interface;
2) Custom launching of an application, executable, or program;
3) Processing the find command locally; or
4) Using MS to MS communications (MS2MS) of FIGS. **75**A and **75**B for remote finding.

In various embodiments, any of the find command Operands can be implemented with either one of the methodologies, although there may be a preference of which methodology is used for which Operand. Atomic find command processing begins at block **6700**, continues to block **6702** for accessing parameters of find command "Operand" (BNF Grammar Operand) and "Parameters" (BNF Grammar Parameters), and then to block **6704** for getting the next (or first) system parameter (block **6704** starts a loop for processing system(s)). At least one system parameter is required for the find. If at least one system is not present for being processed by block **6704**, then block **6704** will handle the error and continue to block **6752** for returning to the caller (not shown—considered obvious error handling, or was already validated at configuration time). Block **6704** continues to block **6706**. If block **6706** determines that an unprocessed system parameter remains, then processing continues to block **6708**. If block **6708** determines the system is not the MS of FIG. **67**A processing, then MS2MS processing is used to accomplish the remote find

processing, in which case block **6708** continues to block **6710** for preparing parameters for FIG. **75**A processing. Thereafter, block **6712** checks to see if there were any parameter errors since block **6710** also validates them prior to preparing them. If block **6712** determines there was at least one parameter error, then block **6713** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing continues back to block **6704**. If block **6713** determines there were no errors, then block **6714** invokes the procedure of FIG. **75**A for sending the data (find command, operand and parameters) for remote find processing at the remote MS. Processing then continues back to block **6704**. MS2MS processing is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs sending data for the find command to the remote MS for finding sought operand dependent criteria at the remote MS, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for the find command. Block **7584** processes the find command for finding sought criteria in context of the Operand at the MS of FIG. **75**B processing. Blocks **7574** and **7576** will return the results to the requesting MS of FIG. **75**A processing, and block **7510** will complete appropriate find processing. Note that block **7510** preferably includes application launch processing (e.g. like found in FIG. **67**A) for invoking the best application in the appropriate manner with the find results returned. The application should be enabled for searching remote MSs further if the user chooses to do so. Another embodiment of block **7510** processes the search results and displays them to the user and/or logs results to a place the user can check later and/or logs results to a place a local MS application can access the results in an optimal manner. In some embodiments, find processing is spawned at the remote MS and the interface results are presented to the remote user. In some embodiments, the find processing results interface is presented to the user of FIG. **67**A processing. In some embodiments, find processing is passed an additional parameter for whether or not to spawn the search interface at the remote MS for the benefit of the remote MS user (at MS of FIG. **75** processing), or to spawn locally for the benefit of the user of the MS of FIG. **67**A processing.

In one embodiment, block **6714** causes processing at a remote data processing system which incorporates similar MS2MS processing, but the remote data processing system is not a MS (i.e. system parameter is for a data processing system identifier accessible to the MS of FIG. **67**A processing). The remote data processing system may be a service data processing system, or any other data processing system capable of similar MS2MS processing as described for the find command, perhaps involving search of storage, memory, or operating system resources which is shared by many MSs.

Referring back to block **6708**, if it is determined that the system for processing is the MS of FIG. **67**A processing, then processing continues to block **6716** for checking which "Operand" was passed. If block **6716** determines the "Operand" indicates to launch a search application for the sought operand with a standard contextual object type interface, then parameter(s) are validated at block **6718** and block **6720** checks the result. If block **6720** determines there was at least one error, then block **6722** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns back to block **6704**. If block **6720** determines there were no parameter errors, then block **6724** interfaces to the MS operating system to start the search application for the particular object passed as a parameter. Block **6724** may prepare parameters in preparation for the O/S contextual launch, for example if parameters are passed to the application which is invoked for finding the object. Processing leaves block **6724** and returns to block **6704**.

An example of block **6724** is similar to the Microsoft Windows XP association of applications to file types for convenient application launch, just as was described above for block **6616**.

Referring back to block **6716**, if it is determined the "Operand" does not indicate to launch with a standard contextual object type interface, processing continues to block **6726**. If block **6726** determines the "Operand" indicates to perform a custom launch, then parameter(s) are validated at block **6728** and block **6730** checks the result. If block **6730** determines there was at least one error, then block **6732** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to block **6704**. If block **6730** determines there were no parameter errors, then processing continues to block **6734**.

If block **6734** determines the custom launch is not to use an Application Programming Interface (API) to launch the applicable search application for finding the object passed as a parameter, then block **6736** prepares a command string for launching the particular application, block **6738** invokes the command string for launching the application, and processing continues to block **6704**.

If block **6734** determines the custom launch is to use an Application Programming Interface (API) to launch the applicable application for finding the object passed as a parameter, then block **6740** prepares any API parameters as necessary, block **6742** invokes the API for launching the application, and processing continues back to block **6704**.

Referring back to block **6726**, if it is determined that the "Operand" indicates to perform the find command with other local processing, then parameter(s) are validated at block **6744** and block **6746** checks the result. If block **6746** determines there was at least one error, then block **6748** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to block **6704**. If block **6748** determines there were no parameter errors, then block **6750** checks the operand for which find processing to perform, and performs find processing appropriately.

Referring back to block **6704**, if it is determined that there are no remaining unprocessed system parameters, then processing returns to the caller at block **6752**.

In FIG. **67**A, "Parameters" for the atomic find command in accordance with the "Operand" were shown to be validated for being properly privileged prior to FIG. **67**A processing (by FIG. **61** processing). However, an alternate embodiment could move some or all applicable privilege validation to FIG. **67**A in context of where the "Parameters" are processed. Also, some embodiments may not validate "Parameters" since they (or some reasonable subset thereof) can be understood to be in good order by the time FIG. **67**A processing occurs (e.g. no blocks **6720**/**6722** and/or **6728**/**6730** and/or **6746**/**6748** required). In yet another embodiment, some defaulting of parameters is implemented.

FIGS. **67**B-**1** through **67**B-**13** depicts a matrix describing how to process some varieties of the Find command (e.g. as processed at blocks **6750** and **7584**). Each row in the matrix describes processing apparatus and/or methods for carrying out command processing for certain operands (see FIG. **34**D for the Operand which matches the number in the first column). The second column shows the Preferred Methodology (PM) for carrying out Find command processing:

S=Standard contextual launch used (blocks **6716** through **6724**);

C=Custom launch used (blocks **6726** through **6742**);

O=Other processing (MS2MS or local) used (blocks **6744** through **6750**, blocks **6708** through **6714**).

Any of the Find command operand combinations can be carried out with either of the methodologies. The second column shows a preferred methodology (PM). The third column describes processing which is placed into flowchart embodiments. There are many embodiments derived from the Find processing descriptions without departing from the spirit and scope of the disclosure. Descriptions are self explanatory.

With reference back to FIGS. **31**A through **31**E, note that the column of information headed by "**107**" represents the parameters applicable for the Find command. The Find command has the following parameters, all of which are interpreted in context of the Operand:

first parameter(s)=These are required, and are in context of the Operand;

system(s)=One or more destination identities for the Find command (e.g. MS ID or a data processing system identifier).

FIG. **67**C depicts a flowchart for describing one embodiment of a procedure for Find command action processing, as derived from the processing of FIG. **67**A. All operands are implemented, and each of blocks F**04** through F**54** can be implemented with any one of the methodologies described with FIG. **67**A, or any one of a blend of methodologies implemented by FIG. **67**C.

Find command processing discussed thus far demonstrates muliithreaded/multiprocessed processing for each system to search. In one embodiment, the same methodology is used for each system and each launched find processing saves results to a common format and destination. In this embodiment, block **6706** processing continues to a new block **6751** when all systems are processed. New block **6751** gathers the superset of find results saved, and then launches an application (perhaps the same one that was launched for each find) to show all results found asynchronously from each other. The application launched will be launched with the same choice of schemes as blocks **6716** through **6750**. Block **6751** then continues to block **6752**. This design requires all applications invoked to terminate themselves after saving search results appropriately for gathering a superset and presenting in one find results interface. Then, the new block **6751** handles processing for a single application to present all search results.

In another embodiment, while an application may be launched multiple times for each system, the application itself is relied upon for handling multiple invocations. The application itself has intelligence to know it was re-launched thereby permitting a single resulting interface for multiple target system searches, regardless of the number of times the same search application was launched.

In one preferred embodiment, find processing permits multiple instances of a search application launched wherein Find processing is treated independently (this is shown in FIG. **67**A).

Preferably all find command embodiments provide the ability to perform other commands (e.g. Copy, Move, Discard, Change, Administrate, etc) wherever possible from the resulting interface in context for each search result found.

Find command data is preferably maintained to LBX history, a historical log, or other useful storage for subsequent use (some embodiments may include this processing where appropriate). Additional find command parameters can be provided for how and where to search (e.g. case sensitivity, get all or first, how to present results, etc).

FIG. **68**A depicts a flowchart for describing a preferred embodiment of a procedure for Invoke command action processing. The Spawn command, Do command, and Invoke command provide identical processing. There are five (5) primary methodologies for carrying out invoke command processing:

1) Launching an application, executable, or program with a standard contextual object type interface;
2) Custom launching of an application, executable, or program;
3) Processing the invoke command locally;
4) Using MS to MS communications (MS2MS) of FIGS. **75**A and **75**B for remote invocation; or
5) Using email or similar messaging layer as a transport layer for invoking distributions.

In various embodiments, any of the invoke command Operands can be implemented with either one of the methodologies, although there may be a preference of which methodology is used for which Operand. Atomic invoke command processing begins at block **6802**, continues to block **6804** for accessing parameters of invoke command "Operand" (BNF Grammar Operand) and "Parameters" (BNF Grammar Parameters), and then to block **6892** for checking if the Operand for invocation indicates to use the email (or similar messaging transport). If block **6892** determines the Operand is for email/messaging transport use, then block **6894** invokes send command processing of FIG. **63**A with the Operand and Parameters. Upon return, processing continues to block **6852** for returning to the caller (invoker of FIG. **68**A processing). If send processing of FIG. **63**A (via block **6894**) is to be used for Operands with a system(s) parameter, then the system(s) parameter is equivalent to the recipient(s) parameter and other parameters are set appropriately.

If block **6892** determines the Operand is not for the email/messaging transport use, then processing continues to block **6806** for getting the next (or first) system parameter (block **6806** starts an iterative loop for processing system(s)). At least one system parameter is required for the invoke command at block **6806**. If at least one system is not present for being processed by block **6806**, then block **6806** will handle the error and continue to block **6852** for returning to the caller (not shown—considered obvious error handling, or was already validated at configuration time). Block **6806** continues to block **6808**. If block **6808** determines that an unprocessed system parameter remains, then processing continues to block **6810**. If block **6810** determines the system is not the MS of FIG. **68**A processing, then MS2MS processing is used to accomplish the remote invoke processing, in which case block **6810** continues to block **6812** for preparing parameters for FIG. **75**A processing, and block **6814** invokes the procedure of FIG. **75**A for sending the data (invoke command, operand and parameters) for remote invoke processing at the remote MS. Processing then continues back to block **6806**. MS2MS processing is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs sending data for the invoke command to the remote MS for an invocation at the remote MS, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for the invoke command. Block **7584** processes the invoke command for invocation in context of the Operand at the MS of FIG. **75** processing (e.g. using invocation methodologies of FIG. **68**A).

In one embodiment, blocks **6812** and **6814** cause processing at a remote data processing system which incorporates similar MS2MS processing, but the remote data processing system is not a MS (i.e. system parameter is for a data processing system identifier accessible to the MS of FIG. **68**A processing). The remote data processing system may be a service data processing system, or any other data processing system capable of similar MS2MS processing as described

for the invoke command, perhaps involving invocation of a suitable executable in context for the operand.

Referring back to block **6810**, if it is determined that the system for processing is the MS of FIG. **68**A processing, then processing continues to block **6816** for checking which "Operand" was passed. If block **6816** determines the "Operand" indicates to invoke (launch) an appropriate application for the operand with a standard contextual object type interface, then parameter(s) are validated at block **6818** and block **6820** checks the result. If block **6820** determines there was at least one error, then block **6822** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns back to block **6806**. If block **6820** determines there were no parameter errors, then block **6824** interfaces to the MS operating system to start the appropriate application for the particular object passed as a parameter. Block **6824** may prepare parameters in preparation for the O/S contextual launch, for example if parameters are passed to the application which is invoked. Processing leaves block **6824** and returns to block **6806**.

An example of block **6824** is similar to the Microsoft Windows XP association of applications to file types for convenient application launch, just as described above for block **6616**.

Referring back to block **6816**, if it is determined the "Operand" does not indicate to launch with a standard contextual object type interface, processing continues to block **6826**. If block **6826** determines the "Operand" indicates to perform a custom launch, then parameter(s) are validated at block **6828** and block **6830** checks the result. If block **6830** determines there was at least one error, then block **6832** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to block **6806**. If block **6830** determines there were no parameter errors, then processing continues to block **6834**.

If block **6834** determines the custom invocation (launch) is not to use an Application Programming Interface (API) to invoke the application for the object passed as a parameter, then block **6836** prepares a command string for invoking the particular application, block **6838** invokes the command string for launching the application, and processing continues to block **6806**.

If block **6834** determines the custom invocation (launch) is to use an Application Programming Interface (API) to invoke the applicable for the object passed as a parameter, then block **6840** prepares any API parameters as necessary, block **6842** invokes the API for launching the application, and processing continues back to block **6806**.

Referring back to block **6826**, if it is determined that the "Operand" indicates to perform the invoke command with other local processing, then parameter(s) are validated at block **6844** and block **6846** checks the result. If block **6846** determines there was at least one error, then block **6848** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to block **6806**. If block **6848** determines there were no parameter errors, then block **6850** checks the operand for which invoke processing to perform, and performs invoke command processing appropriately.

Referring back to block **6808**, if it is determined that there are no remaining unprocessed system parameters, then processing returns to the caller at block **6852**.

In FIG. **68**A, "Parameters" for the atomic invoke command in accordance with the "Operand" were shown to be validated for being properly privileged prior to FIG. **68**A processing (by FIG. **61** processing). However, an alternate embodiment could move some or all applicable privilege validation to FIG. **68**A in context of where the "Parameters" are processed. Also, some embodiments may not validate "Parameters" since they (or some reasonable subset thereof) can be understood to be in good order by the time FIG. **68**A processing occurs (e.g. no blocks **6820**/**6822** and/or **6830**/**6832** and/or **6846**/**6848** required). In yet another embodiment, some defaulting of parameters is implemented.

FIGS. **68**B-**1** through **68**B-**5** depicts a matrix describing how to process some varieties of the Invoke command (e.g. as processed at blocks **6850** and **7584**). Each row in the matrix describes processing apparatus and/or methods for carrying out command processing for certain operands (see FIG. **34**D for the Operand which matches the number in the first column). The second column shows the Preferred Methodology (PM) for carrying out Invoke command processing:

S=Standard contextual launch used (blocks **6816** through **6824**);

C=Custom launch used (blocks **6826** through **6842**);

E=Email transport preferably used (blocks **6892** through **6894**);

O=Other processing (MS2MS or local) used (blocks **6844** through **6850**, blocks **6812** through **6814**).

Any of the Invoke command operand combinations can be carried out with either of the methodologies. The second column shows a preferred methodology (PM). The third column describes processing which is placed into flowchart embodiments. There are many embodiments derived from the Invoke processing descriptions without departing from the spirit and scope of the disclosure. Descriptions are self explanatory.

With reference back to FIGS. **31**A through **31**E, note that the column of information headed by "**109**" represents the parameters applicable for the Invoke command. The Invoke command has the following parameters, all of which are interpreted in context of the Operand:

first parameter(s)=These are required, and are in context of the Operand;

system(s)=One or more destination identities for the Invoke command (e.g. MS ID or a data processing system identifier);

sender=The sender of the Invoke command, typically tied to the originating identity of the action (e.g. email address or MS ID). A different sender can be specified if there is an applicable privilege in place, or if impersonation has been granted;

msg/subj=A message or subject associated with invoke command;

attributes=Indicators for more detailed interpretation of invoke command parameters and/or indicators for attributes to be interpreted by external (e.g. receiving) processes affected by the invoke command result;

recipient(s)=One or more destination identities for the Invoke command (e.g. email address or MS ID).

FIG. **68**C depicts a flowchart for describing one embodiment of a procedure for Invoke command action processing, as derived from the processing of FIG. **68**A. All operands are implemented, and each of blocks J**04** through J**54** can be implemented with any one of the methodologies described with FIG. **68**A, or any one of a blend of methodologies implemented by FIG. **68**C.

In some embodiments, the invoke command may be used as an overall strategy and architecture for performing most, if not all, actions (e.g. other commands).

FIG. **69**A depicts a flowchart for describing a preferred embodiment of a procedure for Copy command action processing. There are four (4) primary methodologies for carrying out copy command search processing:

1) Launching an application, executable, or program with a standard contextual object type interface, for finding the source object(s) to copy;
2) Custom launching of an application, executable, or program, for finding the source object(s) to copy;
3) Processing the copy command locally, for finding the source object(s) to copy; or
4) MS to MS communications (MS2MS) of FIGS. **75**A and **75**B for finding the source object(s) to copy.

The source parameter specifies which system is to be the source of the copy: the MS of FIG. **69**A processing or a remote data processing system.

There are two (2) primary methodologies for carrying out copy command copy processing:

1) Using local processing;
2) MS to MS communications (MS2MS) of FIGS. **75**A and **75**B for remote copying.

In various embodiments, any of the copy command Operands can be implemented with either of the methodologies, although there may be a preference of which methodology is used for which Operand. Atomic copy command processing begins at block **6900**, continues to block **6902** for accessing parameters of copy command "Operand" (BNF Grammar Operand) and "Parameters" (BNF Grammar Parameters), and continues to block **6904**.

If block **6904** determines the source system parameter (source) is this MS, then processing continues to block **6906**. If block **6906** determines the "Operand" indicates to launch a search application for the sought operand object with a standard contextual object type interface, then parameter(s) are validated at block **6908** and block **6910** checks the result. If block **6910** determines there was at least one error, then block **6912** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller (invoker) at block **6960**. If block **6910** determines there were no parameter errors, then block **6914** interfaces to the MS operating system to start the search application for the particular object (for Operand). Block **6914** may prepare parameters in preparation for the operating system. Processing leaves block **6914** and continues to block **6938** which is discussed below.

An example of block **6914** is similar to the Microsoft Windows XP association of applications to file types for convenient application launch, just as was described above for block **6616**.

Referring back to block **6906**, if it is determined the "Operand" does not indicate to launch with a standard contextual object type interface, processing continues to block **6916**. If block **6916** determines the "Operand" indicates to perform a custom launch, then parameter(s) are validated at block **6918** and block **6920** checks the result. If block **6920** determines there was at least one error, then block **6912** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller at block **6960**. If block **6920** determines there were no parameter errors, then processing continues to block **6922**.

If block **6922** determines the custom launch is not to use an Application Programming Interface (API) to launch the searching application for copying the object, then block **6924** prepares a command string for launching the particular application, block **6926** invokes the command string for launching the application, and processing continues to block **6938** discussed below.

If block **6922** determines the custom launch is to use an Application Programming Interface (API) to launch the applicable application for searching, then block **6928** prepares any API parameters as necessary, block **6930** invokes the API for launching the application, and processing continues to block **6938**.

Referring back to block **6916**, if it is determined that the "Operand" indicates to perform the copy command with local search processing, then parameter(s) are validated at block **6932** and block **6934** checks the result. If block **6934** determines there was at least one error, then block **6912** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller at block **6960**. If block **6934** determines there were no parameter errors, then block **6936** searches for the operand object in context for the Operand, and processing continues to block **6938**.

Referring back to block **6904**, if it is determined the source parameter is not for this MS, then block **6962** prepares parameters for FIG. **75**A processing. Thereafter, block **6964** checks to see if there were any parameter errors since block **6962** also validates them prior to preparing them. If block **6764** determines there was at least one parameter error, then block **6712** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller at block **6960**. If block **6764** determines there were no errors, then block **6766** invokes the procedure of FIG. **75**A for sending the data (copy command, operand and parameters) for remote copy search processing at the remote MS. Processing then continues to block **6938** discussed below. MS2MS processing is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs searching for data for the copy command at the remote MS, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for the copy command search processing. Block **7584** processes the copy command for finding the object to copy in context of the Operand. Blocks **7574** and **7576** will return the results to the requesting MS of FIG. **75**A processing, and block **7510** will complete appropriate copy search processing so that FIG. **69**A processing receives the search results. FIG. **75**A can convey the found object(s) for copy by returning from a function interface (the send procedure being a function), returning to a file, setting data visible to both processes, etc. Note that block **7510** may invoke application launch processing (e.g. like found in FIG. **69**A) for invoking the best application in the appropriate manner for determining copy search results returned from FIG. **75**B processing, or block **7510** may process results itself.

In one embodiment, block **6966** causes processing at a remote data processing system which incorporates similar MS2MS processing, but the remote data processing system is not a MS (i.e. system parameter is for a data processing system identifier accessible to the MS of FIG. **67**A processing). The remote data processing system may be a service data processing system, or any other data processing system capable of similar MS2MS processing as described for the find command, perhaps involving search of storage, memory, or operating system resources which are shared by many MSs.

By the time processing reaches block **6938** from any previous FIG. **69**A processing, a search result is communicated to processing and any launched executable (application) for searching for the copy object(s) has terminated. Search results can be passed back as a function return, placed to a well known directory, placed to a file, placed to interfaced variable(s), or other communications of the result to further processing. Regardless of the embodiment, search results are accessed at block **6938**. An alternate embodiment is like FIG. **70**A wherein the application/processing invoked at blocks **6914**, **6926**, **6930** and **6936** handles the ack parameter and ambiguous results appropriately (i.e. no need for blocks **6938**

through **6958**) to proceed with completing the copy (processing of blocks **6938** through **6958** incorporated). Different methods are disclosed for similar processing to highlight methods for carrying out processing for either one of the commands (Copy or Discard).

Block **6938** checks the results of finding the source object for copying to ensure there are no ambiguous results (i.e. not sure what is being copied since the preferred embodiment is to not copy more than a single operand object at a time). If block **6938** determines that there was an ambiguous search result, then processing continues to block **6912** for error handling as discussed above (e.g. in context for an ambiguous copy since there were too many things to copy). If block **6938** determines there is no ambiguous entity to copy, block **6940** checks the acknowledgement parameter passed to FIG. **69**A processing. An alternate embodiment assumes that a plurality of results is valid for copying all results to the target system(s) (i.e. no ambiguous check). In another embodiment, an ambiguous result relies on user reconciliation to reconcile whether or not to perform the copy (like FIG. **70**A discard processing).

If block **6940** determines the acknowledgement (ack) parameter is set to true, then block **6942** provides the search result which is to be copied. Thereafter, processing waits for a user action to either a) continue with the copy; or b) cancel the copy. Once the user action has been detected, processing continues to block **6944**. Block **6942** provides a user reconciliation of whether or not to perform the copy. In another embodiment, there is no ack parameter and multiple results detected at block **6938** forces processing into the reconciliation by the MS user. In yet another embodiment, the ack parameter is still provided, however multiple search results forces processing into the reconciliation by the MS user anyway for selecting which individual object shall be copied. In still other embodiments, all results are copied.

If block **6944** determines the user selected to cancel processing, then block **6946** logs the cancellation (e.g. log error to LBX History **30**) and processing returns to the caller at block **6960**. If block **6944** determines the user selected to proceed with the copy, then processing continues to block **6948** for getting the next (or first) system parameter (block **6948** starts a loop for processing system(s) for the copy result). Also, if block **6940** determines that the ack parameter was set to false, then processing continues directly to block **6948**. At least one system parameter is required for the copy as validated by previous parameter validations. Block **6948** continues to block **6950**. If block **6950** determines that an unprocessed system parameter remains, then processing continues to block **6952**. If block **6952** determines the system (target for copy) is the MS of FIG. **69**A processing, then block **6954** appropriately copies the source object to the system and processing continues back to block **6948**. If block **6952** determines the system is not the MS of FIG. **69**A processing, then MS2MS processing is used to accomplish the copy processing to the remote data processing system (e.g. MS), in which case block **6956** prepares parameters for FIG. **75**A processing, and block **6958** invokes the procedure of FIG. **75**A for sending the data (copy command, operand, and search result) for remote copy processing at the remote MS. Processing then continues back to block **6948**. MS2MS processing is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs sending data for the copy action to the remote MS for copying sought operand dependent criteria to the remote MS, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for the copy processing. Block **7584** processes the copy of the search result from FIG. **69**A to the system of FIG. **75**B processing.

In one embodiment, blocks **6956** and **6958** cause processing at a remote data processing system which incorporates similar MS2MS processing, but the remote data processing system is not a MS (i.e. system parameter is for a data processing system identifier accessible to the MS of FIG. **69**A processing). The remote data processing system may be a service data processing system, or any other data processing system capable of similar MS2MS processing as described for the copy command, perhaps involving storage, memory, or operating system resources which are shared by many MSs.

Referring back to block **6950**, if it is determined that there are no remaining unprocessed system parameters, then processing returns to the caller at block **6960**.

In FIG. **69**A, "Parameters" for the atomic copy command in accordance with the "Operand" were shown to be validated for being properly privileged prior to FIG. **69**A processing (by FIG. **61** processing). However, an alternate embodiment could move some or all applicable privilege validation to FIG. **69**A in context of where the "Parameters" are processed. Also, some embodiments may not validate "Parameters" since they (or some reasonable subset thereof can be understood to be in good order by the time FIG. **69**A processing occurs (e.g. no blocks **6908**/**6910** and/or **6918**/**6920** and/or **6932**/**6934** required). In yet another embodiment, some defaulting of parameters is implemented.

The first parameter may define a plurality of entities to be copied when the object inherently contains a plurality (e.g. directory, container). In an alternate embodiment, the search results for copying can be plural without checking for ambiguity at block **6938**, in which case all results returned can/will be copied to the target systems.

FIGS. **69**B-**1** through **69**B-**14** depicts a matrix describing how to process some varieties of the Copy command. Each row in the matrix describes processing apparatus and/or methods for carrying out command processing for certain operands (see FIG. **34**D for the Operand which matches the number in the first column). The second column shows the Preferred Methodology (PM) for carrying out Copy command processing:

S=Standard contextual launch used (blocks **6906** through **6914**);

C=Custom launch used (blocks **6916** through **6930**);

O=Other processing used (e.g. block **6936**).

Any of the Copy command operand combinations can be carried out with either of the methodologies. The second column shows a preferred methodology (PM). The third column describes processing which is placed into flowchart embodiments. There are many embodiments derived from the Copy processing descriptions without departing from the spirit and scope of the disclosure. Descriptions are self explanatory.

With reference back to FIGS. **31**A through **31**E, note that the column of information headed by "111" represents the parameters applicable for the Copy command. The Copy command has the following parameters, all of which are interpreted in context of the Operand:

first parameter(s)=This is required, and is in context of the Operand;

ack=Boolean for whether or not to prompt user for performing the copy, prior to doing the copy.

source=A source identity for the Copy command (e.g. MS ID or a data processing system identifier);

system(s)=One or more destination identities for the Copy command (e.g. MS ID or a data processing system identifier).

In a preferred embodiment, an additional parameter is provided for specifying the target destination of the system for the copy. For example, a directory can be placed to a target path, an email can be placed to a target folder, etc. Otherwise, there is an assumed target destination. In another embodiment, a user can select from a plurality of search results which objects are to be copied.

FIG. **69**C depicts a flowchart for describing one embodiment of a procedure for Copy command action processing, as derived from the processing of FIG. **69**A. All operands are implemented, and each of blocks C**04** through C**54** can be implemented with any one of the methodologies described with FIG. **69**A, or any one of a blend of methodologies implemented by FIG. **69**C.

FIG. **70**A depicts a flowchart for describing a preferred embodiment of a procedure for Discard command action processing. The Delete command, "Throw Away" command, and Discard command provide identical processing. There are four (4) primary methodologies for carrying out discard command processing:

1) Launching an application, executable, or program with a standard contextual object type interface;
2) Custom launching of an application, executable, or program;
3) Processing the discard command locally; or
4) Using MS to MS communications (MS2MS) of FIGS. **75**A and **75**B for remote discarding.

In various embodiments, any of the discard command Operands can be implemented with either one of the methodologies, although there may be a preference of which methodology is used for which Operand. Atomic discard command processing begins at block **7002**, continues to block **7004** for accessing parameters of discard command "Operand" (BNF Grammar Operand) and "Parameters" (BNF Grammar Parameters), and then to block **7006** for getting the next (or first) system parameter (block **7006** starts an iterative loop for processing system(s)). At least one system parameter is required for the discard. If at least one system is not present for being processed by block **7006**, then block **7006** will handle the error and continue to block **7062** for returning to the caller (not shown—considered obvious error handling, or was already validated at configuration time). Block **7006** continues to block **7008**. If block **7008** determines that an unprocessed system parameter remains, then processing continues to block **7010**. If block **7010** determines the system is not the MS of FIG. **70**A processing, then MS2MS processing is used to accomplish the remote discard processing, in which case block **7010** continues to block **7012** for preparing parameters for FIG. **75**A processing. Thereafter, block **7014** checks to see if there were any parameter errors since block **7012** also validates them prior to preparing them. If block **7014** determines there was at least one parameter error, then block **7016** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing continues back to block **7006**. If block **7014** determines there were no errors, then block **7018** invokes the procedure of FIG. **75**A for sending the data (discard command, operand and parameters) for remote discard processing at the remote MS. Processing then continues back to block **7006**. MS2MS processing is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs sending data for the discard command to the remote MS for discarding sought operand dependent criteria at the remote MS, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for the discard command. Block **7584** processes the discard command for discarding sought criteria in context of the Operand. In a preferred embodiment, the discard takes place when privileged, and when an ack parameter is not provided or is set to false.

Blocks **7574** and **7576** will return the results to the requesting MS of FIG. **75**A processing when the ack parameter is set to true, and block **7510** will complete appropriate discard processing after prompting the user of the MS of FIG. **75**A processing for whether or not to continue ((just like blocks **7054** through **7060** discussed below). Note that block **7510** may include invoking the best application in the appropriate manner (e.g. like found in FIG. **70**A) with the discard results returned when an acknowledgement (ack parameter) has been specified to true, or block **7510** may process results appropriately itself. Processing should be enabled for then continuing with the discard through another invocation of FIG. **75**A (from block **7510** and a following processing of blocks **7578** through **7584** to do the discard) if the user chooses to do so. Block **7510** includes significant processing, all of which has been disclosed in FIG. **70**A anyway and then included at block **7510** if needed there for ack processing.

In one embodiment, block **7018** causes processing at a remote data processing system which incorporates similar MS2MS processing, but the remote data processing system is not a MS (i.e. system parameter is for a data processing system identifier accessible to the MS of FIG. **70**A processing). The remote data processing system may be a service data processing system, or any other data processing system capable of similar MS2MS processing as described for the discard command, perhaps involving search of storage, memory, or operating system resources which are shared by many MSs.

Referring back to block **7010**, if it is determined that the system for processing is the MS of FIG. **70**A processing, then processing continues to block **7020** for checking which "Operand" was passed. If block **7020** determines the "Operand" indicates to launch a search application for the sought operand with a standard contextual object type interface, then parameter(s) are validated at block **7022** and block **7024** checks the result. If block **7024** determines there was at least one error, then block **7016** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns back to block **7006**. If block **7024** determines there were no parameter errors, then block **7026** interfaces to the MS operating system to start the search application for the particular object passed as a parameter and then to continue with the discard for ack set to false, and to prompt for doing the discard for the prompt set to true. Block **7026** may prepare parameters in preparation for the operating system, for example if parameters are passed to the application which is invoked for discarding the object. Processing leaves block **7026** and returns to block **7006**. An alternate embodiment processes like FIG. **69**A wherein the application launched at block **7026** produces only a search result prior to continuing to block **7050**. Then, the search result is discarded if there are no ambiguous results or the ack parameter is set to false, or there are ambiguous results and the user selects to continue, or the ack parameter is set to true and the user selects to continue. FIG. **70**A demonstrates processing where the executable launched is an all inclusive processing. Likewise, FIG. **69**A can be like FIG. **70**A wherein the application launched handles the ack parameter appropriately. Different methods are disclosed for similar processing to highlight methods to carrying out processing for either one of the commands (Copy or Discard).

An example of block **7026** is similar to the Microsoft Windows XP association of applications to file types for convenient application launch, just as was described above for block **6616**.

Referring back to block **7020**, if it is determined the "Operand" does not indicate to launch with a standard contextual object type interface, processing continues to block **7028**. If block **7028** determines the "Operand" indicates to perform a custom launch, then parameter(s) are validated at block **7030** and block **7032** checks the result. If block **7032** determines there was at least one error, then block **7016** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to block **7006**. If block **7032** determines there were no parameter errors, then processing continues to block **7034**.

If block **7034** determines the custom launch is not to use an Application Programming Interface (API) to launch the applicable search application for discarding the object passed as a parameter, then block **7036** prepares a command string for launching the particular application, block **7038** invokes the command string for launching the application, and processing continues to block **7006**. An alternate embodiment processes like FIG. **69**A wherein the application launched at block **7026** produces only a search result prior to continuing to block **7050**. Then, the search result is discarded if there are no ambiguous results or the ack parameter is set to false, or there are ambiguous results and the user selects to continue, or the ack parameter is set to true and the user selects to continue. FIG. **70**A demonstrates processing where the executable launched is an all inclusive processing (e.g. includes processing of blocks **7050** through **7060**).

If block **7034** determines the custom launch is to use an Application Programming Interface (API) to launch the applicable application for discarding the object passed as a parameter, then block **7040** prepares any API parameters as necessary, block **7042** invokes the API for launching the application, and processing continues back to block **7006**. An alternate embodiment processes like FIG. **69**A wherein the application launched at block **7042** produces only a search result prior to continuing to block **7050**. Then, the search result is discarded if there are no ambiguous results or the ack parameter is set to false, or there are ambiguous results and the user selects to continue, or the ack parameter is set to true and the user selects to continue. FIG. **70**A demonstrates processing where the executable launched is an all inclusive processing (includes processing of blocks **7050** through **7060**).

Referring back to block **7028**, if it is determined that the "Operand" indicates to perform the discard command with other local processing, then parameter(s) are validated at block **7044** and block **7046** checks the result. If block **7046** determines there was at least one error, then block **7016** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to block **7006**. If block **7046** determines there were no parameter errors, then block **7048** checks the operand for which discard processing to perform, and performs discard search processing appropriately. Thereafter, block **7050** checks the results.

Block **7050** checks the results of finding the source object for discard to ensure there are no ambiguous results (i.e. not sure what is being discarded since the preferred embodiment is to not discard more than a single operand object at a time). If block **7050** determines that there was an ambiguous search result, then processing continues to block **7052**. If block **7050** determines there is no ambiguity, then processing continues to block **7054**. If block **7054** determines the ack parameter is set to true, then processing continues to block **7052**, otherwise processing continues to block **7060**. Block **7054** checks the acknowledgement parameter passed to FIG. **70**A processing. An alternate embodiment assumes that a plurality of results is valid and discards all results at the target system(s) (i.e. no ambiguous check). In another embodiment, an ambiguous result causes error handling at block **7014** (like FIG. **69**A copy processing).

Block **7052** causes processing for waiting for a user action to either a) continue with the discard; or b) cancel the discard. Once the user action has been detected, processing continues to block **7056**. Block **7052** provides a user reconciliation of whether or not to perform the discard. In another embodiment, there is no ack parameter and multiple results detected at block **7048** are handled for the discard.

If block **7056** determines the user selected to cancel processing, then block **7058** logs the cancellation (e.g. log error to LBX History **30**) and processing returns to block **7006**. If block **7056** determines the user selected to proceed with the discard, then processing continues to block **7060**. Block **7060** performs the discard of the object(s) found at block **7048**. Thereafter, processing continues back to block **7006**.

Referring back to block **7008**, if it is determined that there are no remaining unprocessed system parameters, then processing returns to the caller at block **7062**.

In FIG. **70**A, "Parameters" for the atomic discard command in accordance with the "Operand" were shown to be validated for being properly privileged prior to FIG. **70**A processing (by FIG. **61** processing). However, an alternate embodiment could move some or all applicable privilege validation to FIG. **70**A in context of where the "Parameters" are processed. Also, some embodiments may not validate "Parameters" since they (or some reasonable subset thereof can be understood to be in good order by the time FIG. **70**A processing occurs (e.g. no blocks **7022**/**7024** and/or **7030**/**7032** and/or **7044**/**7046** required). In yet another embodiment, some defaulting of parameters is implemented.

FIGS. **70**B-**1** through **70**B-**11** depicts a matrix describing how to process some varieties of the Discard command. Each row in the matrix describes processing apparatus and/or methods for carrying out command processing for certain operands (see FIG. **34**D for the Operand which matches the number in the first column). The second column shows the Preferred Methodology (PM) for carrying out Discard command processing:

S=Standard contextual launch used (blocks **7020** through **7026**);

C=Custom launch used (blocks **7028** through **7042**);

O=Other processing (MS2MS or local) used (blocks **7044** through **7060**, blocks **7012** through **7018**).

Any of the Discard command operand combinations can be carried out with either of the methodologies. The second column shows a preferred methodology (PM). The third column describes processing which is placed into flowchart embodiments. There are many embodiments derived from the Discard processing descriptions without departing from the spirit and scope of the disclosure. Descriptions are self explanatory.

With reference back to FIGS. **31**A through **31**E, note that the column of information headed by "**113**" represents the parameters applicable for the Discard command. The Discard command has the following parameters, all of which are interpreted in context of the Operand:

first parameter(s)=This is required, and is in context of the Operand;

ack=Boolean for whether or not to prompt user for performing the discard, prior to doing the discard.

system(s)=One or more identities affected for the Discard command (e.g. MS ID or a data processing system identifier).

Discard command processing discussed thus far demonstrates multithreaded/multiprocessed processing for each

system to search. In search results processing, for example when a plurality of results for discard are available, an application may be launched multiple times. For each system, the application itself is relied upon for handling multiple invocations. The application itself has intelligence to know it was re-launched thereby permitting a single resulting interface for multiple target system searches, regardless of the number of times the same search application was launched. In a preferred embodiment, discard processing permits multiple instances of a search application launched. In another embodiment, a user selects which of a plurality of results are to be discarded prior to discarding.

FIG. **70**C depicts a flowchart for describing one embodiment of a procedure for Discard command action processing, as derived from the processing of FIG. **70**A. All operands are implemented, and each of blocks D**04** through D**54** can be implemented with any one of the methodologies described with FIG. **70**A, or any one of a blend of methodologies implemented by FIG. **70**C.

FIG. **71**A depicts a flowchart for describing a preferred embodiment of a procedure for Move command action processing. There are four (4) primary methodologies for carrying out move command search processing:

1) Launching an application, executable, or program with a standard contextual object type interface, for finding the source object(s) to move;
2) Custom launching of an application, executable, or program, for finding the source object(s) to move;
3) Processing the move command locally, for finding the source object(s) to move; or
4) MS to MS communications (MS2MS) of FIGS. **75**A and **75**B for finding the source object(s) to move.

The source parameter specifies which system is to be the source of the move: the MS of FIG. **71**A processing or a remote data processing system.

There are two (2) primary methodologies for carrying out move command processing:

1) Using local processing;
2) MS to MS communications (MS2MS) of FIGS. **75**A and **75**B for remote processing.

In various embodiments, any of the move command Operands can be implemented with either of the methodologies, although there may be a preference of which methodology is used for which Operand. Atomic move command processing begins at block **7100**, continues to block **7102** for accessing parameters of move command "Operand" (BNF Grammar Operand) and "Parameters" (BNF Grammar Parameters), and continues to block **7104**.

If block **7104** determines the source system parameter (source) is this MS, then processing continues to block **7106**. If block **7106** determines the "Operand" indicates to launch a search application for the sought operand object with a standard contextual object type interface, then parameter(s) are validated at block **7108** and block **7110** checks the result. If block **7110** determines there was at least one error, then block **7112** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller (invoker) at block **7160**. If block **7110** determines there were no parameter errors, then block **7114** interfaces to the MS operating system to start the search application for the particular object. Block **7114** may prepare parameters in preparation for the operating system. Processing leaves block **7114** and continues to block **7138** which is discussed below.

An example of block **7114** is similar to the Microsoft Windows XP association of applications to file types for convenient application launch, just as was described above for block **6616**.

Referring back to block **7106**, if it is determined the "Operand" does not indicate to launch with a standard contextual object type interface, processing continues to block **7116**. If block **7116** determines the "Operand" indicates to perform a custom launch, then parameter(s) are validated at block **7118** and block **7120** checks the result. If block **7120** determines there was at least one error, then block **7112** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller at block **7160**. If block **7120** determines there were no parameter errors, then processing continues to block **7122**.

If block **7122** determines the custom launch is not to use an Application Programming Interface (API) to launch the searching application for moving the object, then block **7124** prepares a command string for launching the particular application, block **7126** invokes the command string for launching the application, and processing continues to block **7138** discussed below.

If block **7122** determines the custom launch is to use an Application Programming Interface (API) to launch the applicable application for searching, then block **7128** prepares any API parameters as necessary, block **7130** invokes the API for launching the application, and processing continues to block **7138**.

Referring back to block **7116**, if it is determined that the "Operand" indicates to perform the move command with local search processing, then parameter(s) are validated at block **7132** and block **7134** checks the result. If block **7134** determines there was at least one error, then block **7112** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller at block **7160**. If block **7134** determines there were no parameter errors, then block **7136** searches for the operand object in context for the Operand, and processing continues to block **7138**.

Block **7138** checks the results of finding the source object for moving to ensure there are no ambiguous results (i.e. not sure what is being moved since the preferred embodiment is to not move more than a single operand object at a time). If block **7138** determines there was an ambiguous search result, then processing continues to block **7112** for error handling as discussed above (e.g. in context for an ambiguous move since there were too many things to move). If block **7138** determines there is no ambiguous entity to move, block **7140** checks the acknowledgement parameter passed to FIG. **71**A processing. An alternate embodiment assumes that a plurality of results is valid and moves all results to the target system(s) (i.e. no ambiguous check). In another embodiment, an ambiguous result relies on user reconciliation to reconcile whether or not to perform the move (like FIG. **70**A discard processing).

If block **7140** determines the acknowledgement (ack) parameter is set to true, then block **7142** provides the search result which is to be moved. Thereafter, processing waits for a user action to either a) continue with the move; or b) cancel the move. Once the user action has been detected, processing continues to block **7144**. Block **7142** provides a user reconciliation of whether or not to perform the move. In another embodiment, there is no ack parameter and multiple results detected at block **7138** forces processing into the reconciliation by the user. In yet another embodiment, the ack parameter is still provided, however multiple search results forces processing into the reconciliation by the MS user anyway for selecting which individual object shall be moved. In still other embodiments, all results are moved.

If block **7144** determines the user selected to cancel processing, then block **7146** logs the cancellation (e.g. log error

to LBX History **30**) and processing returns to the caller at block **7160**. If block **7144** determines the user selected to proceed with the move, then processing continues to block **7148** for getting the next (or first) system parameter (block **7148** starts an iterative loop for processing system(s) for the move result). Also, if block **7140** determines that the ack parameter was set to false, then processing continues directly to block **7148**. At least one system parameter is required for the move as validated by previous parameter validations. Block **7148** continues to block **7150**.

If block **7150** determines that an unprocessed system parameter remains, then processing continues to block **7152**. If block **7152** determines the system (target for move) is the MS of FIG. **71**A processing, then block **7154** appropriately moves the source object to the system and processing continues back to block **7148**. If block **7152** determines the system is not the MS of FIG. **71**A processing, then MS2MS processing is used to accomplish the move processing to the remote data processing system (e.g. MS), in which case block **7156** prepares parameters for FIG. **75**A processing, and block **7158** invokes the procedure of FIG. **75**A for sending the data (move command, operand, and search result) for remote move processing at the remote MS. Processing then continues back to block **7148**. MS2MS processing is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs sending data for the move action to the remote MS for moving sought operand dependent criteria to the remote MS, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for the move processing. Block **7584** processes the move of the search result from FIG. **71**A to the system of FIG. **75**B processing.

Referring back to block **7104**, if it is determined the source parameter is not for this MS, then block **7162** prepares parameters for FIG. **75**A processing. Thereafter, block **7164** checks to see if there were any parameter errors since block **7162** also validates them prior to preparing them. If block **7164** determines there was at least one parameter error, then block **7112** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to the caller at block **7160**. If block **7164** determines there were no errors, then block **7166** invokes the procedure of FIG. **75**A for sending the data (move command, operand and parameters) for remote move search processing at the remote MS. Processing then continues to block **7138** discussed below. In one embodiment, the object(s) to move are discarded from the source system (via block **7166**) in preparation for the move command processing at blocks **7154** and **7158**. In another embodiment, the object(s) to move will be discarded from the source system when completing move processing at blocks **7154** or **7158**. MS2MS processing via block **7166** is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs searching for data for the move command at the remote MS, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for at least the move command search processing for the source system. Block **7584** processes the move command for finding the object to move in context of the Operand. Blocks **7574** and **7576** will return the results to the requesting MS of FIG. **75**A processing, and block **7510** will complete appropriate move search processing so that FIG. **71**A processing receives the search results. FIG. **75**A can convey the found object(s) for the move by returning from a function interface (the send procedure being a function), returning to a file, setting data visible to both processes, etc. Note that block **7510** may include application launch processing (e.g. like found in FIG. **71**A) for invoking the best application in the appropriate manner for determining move search results returned from FIG. **75**B processing, or block **7510** may process returned results itself.

In one embodiment, block **7166** causes processing at a remote data processing system which incorporates similar MS2MS processing, but the remote data processing system is not a MS (i.e. system parameter is for a data processing system identifier accessible to the MS of FIG. **71**A processing). The remote data processing system may be a service data processing system, or any other data processing system capable of similar MS2MS processing as described for the find command, perhaps involving search of storage, memory, or operating system resources which are shared by many MSs.

By the time processing reaches block **7138** from any previous FIG. **71**A processing, a search result is communicated to processing and any launched executable (application) for searching for the move object(s) has terminated. Search results can be passed back as a function return, placed to a well known directory, placed to a file, placed to interfaced variable(s), or other communications of the result to further processing. Regardless of the embodiment, search results are accessed at block **7138**. An alternate embodiment is like FIG. **70**A wherein the application/processing invoked at blocks **7114**, **7126**, **7130** and **7136** handles the ack parameter and ambiguous results appropriately (i.e. no need for blocks **7138** through **7158**) to proceed with completing the move (processing of blocks **7138** through **7158** incorporated). Different methods are disclosed for similar processing to highlight methods for carrying out processing for either one of the commands (Move or Discard).

In one embodiment, blocks **7156** and **7158** cause processing at a remote data processing system which incorporates similar MS2MS processing, but the remote data processing system is not a MS (i.e. system parameter is for a data processing system identifier accessible to the MS of FIG. **71**A processing). The remote data processing system may be a service data processing system, or any other data processing system capable of similar MS2MS processing as described for the move command, perhaps involving storage, memory, or operating system resources which are shared by many MSs.

Referring back to block **7150**, if it is determined that there are no remaining unprocessed system parameters, then processing returns to the caller at block **7160**.

In FIG. **71**A, "Parameters" for the atomic move command in accordance with the "Operand" were shown to be validated for being properly privileged prior to FIG. **71**A processing (by FIG. **61** processing). However, an alternate embodiment could move some or all applicable privilege validation to FIG. **71**A in context of where the "Parameters" are processed. Also, some embodiments may not validate "Parameters" since they (or some reasonable subset thereof) can be understood to be in good order by the time FIG. **71**A processing occurs (e.g. no blocks **7108**/**7110** and/or **7118**/**7120** and/or **7132**/**7134** required). In yet another embodiment, some defaulting of parameters is implemented.

The first parameter may define a plurality of entities to be moved when the object inherently contains a plurality (e.g. directory, container). In an alternate embodiment, the search results for moving can be plural without checking for ambiguity at block **7138**, in which case all results returned will be moved to the target systems.

FIGS. **71**B-**1** through **71**B-**14** depicts a matrix describing how to process some varieties of the Move command. The end result of a move command is identical to "Copy" command processing except the source is "Discard"-ed as part of processing (preferably after the copy). Each row in the matrix

describes processing apparatus and/or methods for carrying out command processing for certain operands (see FIG. **34**D for the Operand which matches the number in the first column). The second column shows the Preferred Methodology (PM) for carrying out Move command processing:

S=Standard contextual launch used (blocks **7106** through **7114**);

C=Custom launch used (blocks **7116** through **7130**);

O=Other processing used (e.g. block **7136**).

Any of the Move command operand combinations can be carried out with either of the methodologies. The second column shows a preferred methodology (PM). The third column describes processing which is placed into flowchart embodiments. There are many embodiments derived from the Move processing descriptions without departing from the spirit and scope of the disclosure. Descriptions are self explanatory.

With reference back to FIGS. **31**A through **31**E, note that the column of information headed by "**115**" represents the parameters applicable for the Move command. The Move command has the following parameters, all of which are interpreted in context of the Operand:

first parameter(s)=This is required, and is in context of the Operand;

ack=Boolean for whether or not to prompt user for performing the move, prior to doing the move.

source=A source identity for the Move command (e.g. MS ID or a data processing system identifier);

system(s)=One or more destination identities for the Move command (e.g. MS ID or a data processing system identifier).

In an alternate embodiment, an additional parameter is provided for specifying the target destination of the system for the move. For example, a directory can be placed to a target path, an email can be placed to a target folder, etc.

FIG. **71**C depicts a flowchart for describing one embodiment of a procedure for Move command action processing, as derived from the processing of FIG. **71**A. All operands are implemented, and each of blocks M**04** through M**54** can be implemented with any one of the methodologies described with FIG. **71**A, or any one of a blend of methodologies implemented by FIG. **71**C.

FIG. **72**A depicts a flowchart for describing a preferred embodiment of a procedure for Store command action processing. There are four (4) primary methodologies for carrying out store command processing:

1) Launching an application, executable, or program with a standard contextual object type interface;
2) Custom launching of an application, executable, or program;
3) Processing the store command locally; or
4) Using MS to MS communications (MS2MS) of FIGS. **75**A and **75**B for storing remotely.

In various embodiments, any of the store command Operands can be implemented with either one of the methodologies, although there may be a preference of which methodology is used for which Operand. Atomic store command processing begins at block **7202**, continues to block **7204** for accessing parameters of store command "Operand" (BNF Grammar Operand) and "Parameters" (BNF Grammar Parameters), and then to block **7206** for getting the next (or first) system parameter (block **7206** starts an iterative loop for processing system(s)). At least one system parameter is required for the store command. If at least one system is not present for being processed by block **7206**, then block **7206** will handle the error and continue to block **7250** for returning to the caller (not shown—considered obvious error handling, or was already validated at configuration time). Block **7206** continues to block **7208**. If block **7208** determines that an unprocessed system parameter remains, then processing continues to block **7210**. If block **7210** determines the system is not the MS of FIG. **72**A processing, then MS2MS processing is needed to accomplish the remote store processing, in which case block **7210** continues to block **7212** for preparing parameters for FIG. **75**A processing. Thereafter, block **7214** checks to see if there were any parameter errors since block **7212** also validates them prior to preparing them. If block **7214** determines there was at least one parameter error, then block **7216** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing continues back to block **7206**. If block **7214** determines there were no errors, then block **7218** invokes the procedure of FIG. **75**A for sending the data (store command, operand and parameters) for remote store processing at the remote MS. Processing then continues back to block **7206**. MS2MS processing is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs sending data for the store command to the remote MS for storing operand dependent criteria at the remote MS, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for the store command. Block **7584** processes the store command for storing in context of the Operand.

In one embodiment, block **7218** causes processing at a remote data processing system which incorporates similar MS2MS processing, but the remote data processing system is not a MS (i.e. system parameter is for a data processing system identifier accessible to the MS of FIG. **72**A processing). The remote data processing system may be a service data processing system, or any other data processing system capable of similar MS2MS processing as described for the store command, perhaps involving search of storage, memory, or operating system resources which are shared by many MSs.

Referring back to block **7208**, if it is determined that the system for processing is the MS of FIG. **72**A processing, then processing continues to block **7220** for checking which "Operand" was passed. If block **7220** determines the "Operand" indicates to launch a store application for the sought operand with a standard contextual object type interface, then parameter(s) are validated at block **7222** and block **7224** checks the result. If block **7224** determines there was at least one error, then block **7216** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns back to block **7206**. If block **7224** determines there were no parameter errors, then block **7226** interfaces to the MS operating system to start the storing application for the particular object passed as a parameter. Block **7226** may prepare parameters in preparation for the operating system, for example if parameters are passed to the application which is invoked for storing the object. Processing leaves block **7226** and returns to block **7206**.

An example of block **7226** is similar to the Microsoft Windows XP association of applications to file types for convenient application launch, just as was described above for block **6616**.

Referring back to block **7220**, if it is determined the "Operand" does not indicate to launch with a standard contextual object type interface, processing continues to block **7228**. If block **7228** determines the "Operand" indicates to perform a custom launch, then parameter(s) are validated at block **7230** and block **7232** checks the result. If block **7232** determines there was at least one error, then block **7216** handles the error appropriately (e.g. log error to LBX History **30** and/or notify

user) and processing returns to block **7206**. If block **7232** determines there were no parameter errors, then processing continues to block **7234**.

If block **7234** determines the custom launch is not to use an Application Programming Interface (API) to launch the applicable application for storing the object passed as a parameter, then block **7236** prepares a command string for launching the particular application, block **7238** invokes the command string for launching the application, and processing continues to block **7206**.

If block **7234** determines the custom launch is to use an Application Programming Interface (API) to launch the applicable application for storing the object passed as a parameter, then block **7240** prepares any API parameters as necessary, block **7242** invokes the API for launching the application, and processing continues back to block **7206**.

Referring back to block **7228**, if it is determined that the "Operand" indicates to perform the store command with other local processing, then parameter(s) are validated at block **7244** and block **7246** checks the result. If block **7246** determines there was at least one error, then block **7216** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to block **7206**. If block **7246** determines there were no parameter errors, then block **7248** checks the operand for which store processing to perform, and performs store processing appropriately.

Referring back to block **7206**, if it is determined that there are no remaining unprocessed system parameters, then processing returns to the caller at block **7250**.

In FIG. **72**A, "Parameters" for the atomic store command in accordance with the "Operand" were shown to be validated for being properly privileged prior to FIG. **72**A processing (by FIG. **61** processing). However, an alternate embodiment could move some or all applicable privilege validation to FIG. **72**A in context of where the "Parameters" are processed. Also, some embodiments may not validate "Parameters" since they (or some reasonable subset thereof) can be understood to be in good order by the time FIG. **72**A processing occurs (e.g. no blocks **7222**/**7224** and/or **7230**/**7232** and/or **7244**/**7246** required). In yet another embodiment, some defaulting of parameters is implemented.

FIGS. **72**B-**1** through **72**B-**5** depicts a matrix describing how to process some varieties of the Store command. Each row in the matrix describes processing apparatus and/or methods for carrying out command processing for certain operands (see FIG. **34**D for the Operand which matches the number in the first column). The second column shows the Preferred Methodology (PM) for carrying out Store command processing:

S=Standard contextual launch used (blocks **7220** through **7226**);

C=Custom launch used (blocks **7228** through **7242**);

O=Other processing (MS2MS or local) used (blocks **7244** through **7248**, blocks **7212** through **7218**).

Any of the Store command operand combinations can be carried out with either of the methodologies. The second column shows a preferred methodology (PM). The third column describes processing which is placed into flowchart embodiments. There are many embodiments derived from the Store processing descriptions without departing from the spirit and scope of the disclosure. Descriptions are self explanatory.

With reference back to FIGS. **31**A through **31**E, note that the column of information headed by "**117**" represents the parameters applicable for the Store command. The Store command has the following parameters, all of which are interpreted in context of the Operand:

first parameter(s)=These are required, and are in context of the Operand;

system(s)=One or more destination identities for the Store command (e.g. MS ID or a data processing system identifier).

In an alternate embodiment, an ack parameter is provided for proving a user reconciliation of the store processing (like ack parameter in other commands) wherein the reconciliation preferably presents the proposed store operation in an informative manner so that the user can make an easy decision to proceed or cancel.

FIG. **72**C depicts a flowchart for describing one embodiment of a procedure for Store command action processing, as derived from the processing of FIG. **72**A. All operands are implemented, and each of blocks R**04** through R**54** can be implemented with any one of the methodologies described with FIG. **72**A, or any one of a blend of methodologies implemented by FIG. **72**C.

FIG. **73**A depicts a flowchart for describing a preferred embodiment of a procedure for Administrate command action processing. There are four (4) primary methodologies for carrying out administrate command processing:

1) Launching an application, executable, or program with a standard contextual object type interface;
2) Custom launching of an application, executable, or program;
3) Processing the administrate command locally; or
4) Using MS to MS communications (MS2MS) of FIGS. **75**A and **75**B for remote administration.

In various embodiments, any of the administrate command Operands can be implemented with either one of the methodologies, although there may be a preference of which methodology is used for which Operand. Atomic administrate command processing begins at block **7302**, continues to block **7304** for accessing parameters of administrate command "Operand" (BNF Grammar Operand) and "Parameters" (BNF Grammar Parameters), and then to block **7306** for getting the next (or first) system parameter (block **7306** starts an iterative loop for processing system(s)). At least one system parameter is required for the administrate command. If at least one system is not present for being processed by block **7306**, then block **7306** will handle the error and continue to block **7350** for returning to the caller (not shown—considered obvious error handling, or was already validated at configuration time). Block **7306** continues to block **7308**. If block **7308** determines that an unprocessed system parameter remains, then processing continues to block **7310**. If block **7310** determines the system is not the MS of FIG. **73**A processing, then MS2MS processing is needed to accomplish the remote administration processing, in which case block **7310** continues to block **7312** for preparing parameters for FIG. **75**A processing. Thereafter, block **7314** checks to see if there were any parameter errors since block **7312** also validates them prior to preparing them. If block **7314** determines there was at least one parameter error, then block **7316** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing continues back to block **7306**. If block **7314** determines there were no errors, then block **7318** invokes the procedure of FIG. **75**A for sending the data (administrate command, operand and parameters) for remote administrate processing at the remote MS. Processing then continues back to block **7306**. MS2MS processing is as already described above (see FIGS. **75**A and **75**B), except FIG. **75**A performs sending data for the administrate command to the remote MS for searching for sought operand dependent criteria at the remote MS, and FIG. **75**B blocks **7578** through **7584** carry out processing specifically for the

administrate command search result. Block **7584** processes the administrate command for searching for sought criteria in context of the Operand. Blocks **7574** and **7576** will return the results to the requesting MS of FIG. **75**A processing, and block **7510** will complete appropriate administrate processing. Note that block **7510** may include application launch processing (e.g. like found in FIG. **73**A) for invoking the best application in the appropriate manner with the administrate results returned. The application should be enabled for searching remote MSs further if the user chooses to do so, and be enabled to perform the privileged administration. Another embodiment of block **7510** processes the search results and displays them to the user for subsequent administration in an optimal manner. In some embodiments, administrate processing is spawned at the remote MS and the interface results are presented to the remote user. In preferred embodiments, the administrate processing results interface is presented to the user of FIG. **73**A processing for subsequent administration. In some embodiments, administrate processing is passed an additional parameter for whether or not to spawn the search interface at the remote MS for the benefit of the remote MS user, or to spawn locally for the benefit of the user of the MS of FIG. **73**A processing. Block **7510** may process results itself.

In one embodiment, block **7318** causes processing at a remote data processing system which incorporates similar MS2MS processing, but the remote data processing system is not a MS (i.e. system parameter is for a data processing system identifier accessible to the MS of FIG. **73**A processing). The remote data processing system may be a service data processing system, or any other data processing system capable of similar MS2MS processing as described for the administrate command, perhaps involving search of storage, memory, or operating system resources which are shared by many MSs.

Referring back to block **7310**, if it is determined that the system for processing is the MS of FIG. **73**A processing, then processing continues to block **7320** for checking which "Operand" was passed. If block **7320** determines the "Operand" indicates to launch the administration application for the sought operand with a standard contextual object type interface, then parameter(s) are validated at block **7322** and block **7324** checks the result. If block **7324** determines there was at least one error, then block **7316** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns back to block **7306**. If block **7324** determines there were no parameter errors, then block **7326** interfaces to the MS operating system to start the administration application for the particular object passed as a parameter. Block **7326** may prepare parameters in preparation for the operating system, for example if parameters are passed to the application which is invoked for administration of the object. Processing leaves block **7326** and returns to block **7306**.

An example of block **7326** is similar to the Microsoft Windows XP association of applications to file types for convenient application launch, just as was described above for block **6616**.

Referring back to block **7320**, if it is determined the "Operand" does not indicate to launch with a standard contextual object type interface, processing continues to block **7328**. If block **7328** determines the "Operand" indicates to perform a custom launch, then parameter(s) are validated at block **7330** and block **7332** checks the result. If block **7332** determines there was at least one error, then block **7316** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to block **7306**. If block **7332** determines there were no parameter errors, then processing continues to block **7334**.

If block **7334** determines the custom launch is not to use an Application Programming Interface (API) to launch the applicable administration application for administration of the object passed as a parameter, then block **7336** prepares a command string for launching the particular application, block **7338** invokes the command string for launching the application, and processing continues to block **7306**.

If block **7334** determines the custom launch is to use an Application Programming Interface (API) to launch the applicable application for administration of the object passed as a parameter, then block **7340** prepares any API parameters as necessary, block **7342** invokes the API for launching the application, and processing continues back to block **7306**.

Referring back to block **7328**, if it is determined that the "Operand" indicates to perform the administrate command with other local processing, then parameter(s) are validated at block **7344** and block **7346** checks the result. If block **7346** determines there was at least one error, then block **7316** handles the error appropriately (e.g. log error to LBX History **30** and/or notify user) and processing returns to block **7306**. If block **7346** determines there were no parameter errors, then block **7348** checks the operand for which administration processing to perform, and performs administration processing appropriately.

Referring back to block **7306**, if it is determined that there are no remaining unprocessed system parameters, then processing returns to the caller at block **7350**.

In FIG. **73**A, "Parameters" for the atomic administrate command in accordance with the "Operand" were shown to be validated for being properly privileged prior to FIG. **73**A processing (by FIG. **61** processing). However, an alternate embodiment could move some or all applicable privilege validation to FIG. **73**A in context of where the "Parameters" are processed. Also, some embodiments may not validate "Parameters" since they (or some reasonable subset thereof) can be understood to be in good order by the time FIG. **73**A processing occurs (e.g. no blocks **7322**/**7324** and/or **7330**/**7332** and/or **7344**/**7346** required). In yet another embodiment, some defaulting of parameters is implemented.

FIGS. **73**B-**1** through **73**B-**7** depicts a matrix describing how to process some varieties of the Administrate command. Each row in the matrix describes processing apparatus and/or methods for carrying out command processing for certain operands (see FIG. **34**D for the Operand which matches the number in the first column). The second column shows the Preferred Methodology (PM) for carrying out Administrate command processing:

S=Standard contextual launch used (blocks **7320** through **7326**);

C=Custom launch used (blocks **7328** through **7342**);

O=Other processing (MS2MS or local) used (blocks **7344** through **7348**, blocks **7308** through **7318**).

Any of the Administrate command operand combinations can be carried out with either of the methodologies. The second column shows a preferred methodology (PM). The third column describes processing which is placed into flowchart embodiments. There are many embodiments derived from the Administrate processing descriptions without departing from the spirit and scope of the disclosure. Descriptions are self explanatory.

With reference back to FIGS. **31**A through **31**E, note that the column of information headed by "**121**" is not shown. However, it is assumed to be present (. . .). The Administrate

command has the following parameters, all of which are interpreted in context of the Operand:

first parameter(s)=These are required, and are in context of the Operand;

system(s)=One or more destination identities for the Administrate command (e.g. MS ID or a data processing system identifier).

FIG. **73**C depicts a flowchart for describing one embodiment of a procedure for Administrate command action processing, as derived from the processing of FIG. **73**A. All operands are implemented, and each of blocks A**04** through A**54** can be implemented with any one of the methodologies described with FIG. **73**A, or any one of a blend of methodologies implemented by FIG. **73**C.

Administrate command processing discussed thus far demonstrates multithreaded/multiprocessed processing for each system to perform administration. In one embodiment, the same methodology is used for each system and each launched administrate processing saves results to a common format and destination. In this embodiment, block **7308** processing continues to a new block **7349** when all systems are processed. New block **7349** gathers the superset of administrate results saved, and then launches an application (perhaps the same one that was launched for each administrate) to show all results found asynchronously from each other. The application launched will be launched with the same choice of schemes as blocks **7320** through **7350**. Block **7349** then continues to block **7350**. This design will want all applications invoked to terminate themselves after saving search results appropriately. Then, the new block **7349** starts a single administration application to present all search results for performing the administration.

In another embodiment, while an application may be launched multiple times for each system, the application itself is relied upon for handling multiple invocations. The application itself has intelligence to know it was re-launched thereby permitting a single resulting interface for multiple target system searches, regardless of the number of times the same search application was launched.

In one preferred embodiment, administrate processing permits multiple instances of a search application launched. Administrate processing is treated independently (this is shown in FIG. **73**A).

Preferably all administrate command embodiments provide the ability to perform other commands (e.g. Copy, Move, Discard, Change, . . .) wherever possible from the resulting interface in context for each search result found.

There are many other reasonable commands (and operands), some of which may intersect processing by other commands. For example, there is a change command. The change command can be described by operand as the other commands were, except the change command has identical processing to other commands for a particular operand. There are multiple commands duplicated with the change command, depending on the operand of the change command (like Connect command overlap of functionality). FIG. **74**A depicts a flowchart for describing a preferred embodiment of a procedure for Change command action processing, and FIG. **74**C depicts a flowchart for describing one embodiment of a procedure for Change command action processing, as derived from the processing of FIG. **74**A.

Charters certainly provide means for a full spectrum of automated actions from simple predicate based (conditional) alerts to complex application processing. Actions includes API invocations, executable script invocations (e.g. from command line), executable program invocations, O/S contextual launch executions, integrated execution processing (e.g. part of block processing), or any other processing executions. As incoming WDRs indicate that a MS (MS user) of interest is nearby, charters provide the mechanism for the richest possible executions of many varieties to be automatically processed. From as simple a use as generating nearby/nearness/distantness status to performing a complicated set of processing based on nearby/nearness/distantness relative a MS user, there is no limit to the processing that can occur. All of the processing is handled locally by the MS and no connected service was required.

A first LBX enabled MS with phone capability can have a charter configuration for automatically placing a call to a second LBX enabled MS user upon determining that the second MS is close by the first MS user, for example when both users are coincidentally nearby each other. Perhaps the users are in a store at the same time, or are attending an event without knowledge of each other's attendance. It is "cool" to be able to cause an automatic phone call for connecting the users by conversation to then determine that they should "hook up" since they are nearby. Furthermore, a charter at the first MS can be configured wherein the first MS automatically dials/calls the second MS user, or alternatively a charter at the first MS can be configured wherein the second MS automatically dials/calls the first MS user, provided appropriate privileges are in place.

FIG. **76** depicts a flowchart for describing a preferred embodiment of processing a special Term (BNF Grammar Term: WDRTerm, AppTerm, atomic term, etc) information paste action at a MS. Special paste action processing begins at block **7602** upon detection of a user invoked action to perform a special paste using Term information. Depending on the embodiment, FIG. **76** processing is integrated into the MS user interface processing, either as presentation manager code, a plug-in, TSR (Terminate and Stay Resident) code, or other method for detecting applicable user input at the MS (e.g. keystroke(s), voice command, etc). Unique paste requests (user actions) cause processing to start at block **7602**. Block **7602** continues to block **7604** where the most recent Term information for the MS of FIG. **76** processing is accessed, then to block **7606** to see if the referenced value for the paste is set. Depending on when a user invokes the special paste option, the sought Term for pasting may not have a value set yet (e.g. AppTerm newly registered). If block **7606** determines the Term has not yet been set with a value, then block **7608** default the value for paste, otherwise block **7606** continues to block **7610**. Block **7608** may or may not choose to default with an obvious value for "not set yet". If block **7610** determines the Term to be pasted is a WDRTerm, then processing continues to block **7612** where the WTV is accessed, and then to block **7614** to see how timely the most recent WDR accessed at block **7604** is for describing whereabouts of the MS. If block **7614** determines the WDR information is not out of date with respect to the WTV (i.e. whereabouts information is timely), then block **7616** pastes the WDR information according to the special paste action causing execution of FIG. **76**. If there is no data entry field in focus at the MS at the time of FIG. **76** processing, then an error occurs at block **7616** which is checked for at block **7618**. If block **7618** determines the WDR information paste operation was successful, processing terminates at block **7622**, otherwise block **7620** provides the user with an error that there is no data entry field in focus applicable for the paste operation. The error may require a user acknowledgement to clear the error to ensure the user sees the error. Block **7620** then continues to block **7622**.

If at block **7614** it is determined the user attempted to paste WDR information from an untimely WDR, then block **7624**

provides the user with a warning, preferably including how stale the WDR information is, and processing waits for a user action to proceed with the paste, or cancel the paste. Thereafter, if block **7626** determines the user selected to cancel the paste operation, then processing terminates at block **7622**, otherwise processing continues to block **7616**.

Referring back to block **7610**, if it determined the paste operation is not for a WDRTerm, then processing continues directly to block **7616** for pasting the other Term construct terms being referenced by the paste operation (i.e. atomic term, AppTerm, etc).

FIG. **76** processes special paste commands for pasting Term information to data entry fields of the MS user interface from Term data maintained at the MS. In a preferred embodiment, queue **22** is accessed for the most recent WDR at block **7604** when a WDRTerm (WDR field/subfield) is referenced. In another embodiment, a single WDR entry for the most recent WDR information is accessed at block **7604**. In a preferred embodiment, there are a plurality of special paste commands detected and each command causes pasting the associated Term information field(s) in an appropriate format to the currently focused user interface data entry field. There can be a command (user input) for pasting any Term (e.g. WDR) field(s) in a particular format to the currently focused data entry field. In another embodiment, one or more fields are accessed at block **7616** and then used to determine an appropriate content for the paste operation to the currently focused data entry field. For example, there can be a special keystroke sequence (<Ctrl><Alt><l>) to paste a current location (e.g. WDRTerm WDR field **1100***c*) to the currently focused data entry field, a special keystroke sequence (<Ctrl><Alt><s>) to paste a current situational location to the currently focused data entry field (e.g. my most recent atomic term situational location), a special keystroke sequence (<Ctrl><Alt><i>) to paste the MS ID of the most recently received WDR, a special keystroke sequence (<Ctrl><Alt><c>) to paste a confidence (e.g. WDRTerm WDR field **1100***d*) to the currently focused data entry field, a special keystroke sequence (<Ctrl><Alt><e>) to paste a current email source address from the WDR application fields section of the WDR, a special keystroke sequence (<Ctrl><Alt><F1>) to paste a current email source address from the WDR application fields section of the WDR, a special keystroke sequence (<Ctrl><Alt><1>) to paste a current statistical atomic term, etc. There can be a user input for pasting any Term data including from WDRs, atomic terms (Value construct), Application Terms, most recent Invocation, etc.

In another embodiment, the keystroke sequence for the particular paste operation includes a keystroke as defined in a prefix **5300***a*, or in a new record field **5300***i* for an application, so that particular application field(s) are accessible from WDR Application fields **1100***k*. In other embodiments, there are special paste actions for LBX maintained statistics, whereabouts information averages, or any other useful current or past LBX data, including from LBX History **30**. In another embodiment, there are special paste actions for predicted data which is based on current and/or passed LBX data, for example using an automated analysis of a plurality of WDRs, application terms, atomic terms, statistics, or information thereof.

Application Fields **1100***k*

Application fields **1100***k* are preferably set in a WDR when it is completed for queue **22** insertion (for FIG. **2**F processing). This ensures WDRs which are in-process to queue **22** contain the information at appropriate times. This also ensures the WDRs which are to be sent outbound contain the information at the appropriate time, and ensures the WDRs which are to be received inbound contain the information at the appropriate time. Fields **1100***k* may be set when processing at inbound time as well. Application fields can add a significant amount of storage to a WDR. Alternate embodiments may not maintain field **1100***k* to queue **22**, but rather append information, or an appropriate subset thereof, to field **1100***k* when sending WDRs outbound to minimize storage WDRs utilize at a MS. This alternate embodiment will enable appropriate WITS processing for maintained WDRs, inbound WDRs, and outbound WDRs without an overhead of maintaining lots of data to queue **22**, however application fields functionality will be limited to application data from an outbound originated perspective, rather than application field setting at the time of an in process WDR regardless of when it was in process. For example, field **1100***k* may alternatively be set at blocks **2014** and **2514** and then stripped after being processed by receiving MSs prior to any insertion to queue **22**. In some embodiments, certain field **1100***k* data can be enabled or disabled for being present in WDR information.

Preferably, there are WDRTerms for referencing each reasonable application fields section individually, as a subset, or as a set. For example, _appfid.appname.dataitem should resolve to the value of "dataitem" for the application section "appname" of application fields **1100***k* (i.e. "_appfld"). The hierarchy qualification operator (i.e. ".") indicates which subordinate member is being referenced for which organization is use of field **1100***k*. The requirement is the organization be consistent in the LN-expanse (e.g. data values for anticipated application categories). For example, _appfld.email.source resolves to the email address associated with the email application of the MS which originated the WDR. For example, _appfld.phone.id resolves to the phone number associated with the phone application of the MS which originated the WDR (e.g. for embodiments where the MS ID is not the same as the MS caller id/phone number). If a WDRTerm references an application field which is not present in a WDR, then preferably a run time error during WITS processing is logged with ignoring of the expression and any assigned action, or the applicable condition defaults to false. Preferably, a user has control for enabling any application subsets of data in field **1100***k*.

FIG. **77** depicts a flowchart for describing a preferred embodiment of configuring data to be maintained to WDR Application Fields **1100***k*. While there can certainly be privileges put in place to govern whether or not to include certain data in field **1100***k*, it may be desirable to differentiate this because of the potentially large amount of storage required to carry such data when transmitting and processing WDRs. Highlighting such consideration and perhaps warning a user of its use may be warranted. FIG. **72** processing provides the differentiation. Depending on present disclosure implementations, there are privileges which require associated information, for example for enabling profile communication (preferably can define which file is to be used for the profile), accepting data/database/file control (preferably can define which data and what to do), etc. An alternate embodiment may define a specific privilege for every derivation, but this may overwhelm a user when already configuring many privileges. Also, specific methods may be enforced without allowing user specification (e.g. always use a certain file for the profile). A preferred embodiment permits certain related specifications with privileges and also differentiates handling of certain features which could be accomplished with privileges.

Application fields **1100**K specification processing begins at block **7702** upon a user action for the user interface processing of FIG. **77**, and continues to block **7704** where the user is presented with options. Thereafter, block **7706** waits for a user input/action. The user is able to specify any of a plurality of application data for enablement or disablement in at least outbound WDR fields **1100***k*. Various embodiments will support enablement/disablement for inbound, outbound, or any other in-process WDR event executable processing paths. Field **1100***k* can be viewed as containing application sections, each section containing data for a particular type of MS application, or a particular type of application data as described above.

Upon detection of a user action at block **7706**, block **7708** checks if the user selected to enable a particular application section of fields **1100***k*. If block **7708** determines the user selected to enable a particular application fields **1100***k* section, then block **7710** sets the particular indicator for enabling that particular application fields **1100***k* section, and processing continues back to block **7704**. If block **7708** determines the user did not select to enable a particular application fields **1100***k* section, then processing continues to block **7712**. If block **7712** determines the user selected to disable a particular application fields **1100***k* section, then block **7714** sets the particular indicator for disabling that particular application fields **1100***k* section, and processing continues back to block **7704**. If block **7712** determines the user did not select to disable a particular application fields **1100***k* section, then processing continues to block **7716**. If block **7716** determines the user selected to disable sending profile information in a application fields **1100***k* section, then block **7718** sets the profile participation variable to NULL (i.e. disabled), and processing continues back to block **7704**. If block **7716** determines the user did not select to disable sending profile information, then processing continues to block **7720**. If block **7720** determines the user selected to enable sending profile information in a application fields **1100***k* section, then block **7722** prompts the user for the file to be used for the profile (preferably the last used (or best used) file is defaulted in the interface), and block **7724** interfaces with the user for a validated file path specification. The user may not be able to specify a validated profile specification at block **7724** in which case the user can cancel out of block **7724** processing. Thereafter, if block **7726** determines the user cancelled out of block **7724** processing, processing continues back to block **7704**. If block **7726** determines the user specified a validated profile file, then block **7728** sets the profile participation variable to the fully qualified path name of the profile file, and processing continues back to block **7704**. Block **7724** preferably parses the profile to ensure it conforms to an LN-expanse standard format, or error processing is handled which prevents the user from leaving block **7724** with an incorrect profile.

In an alternate embodiment, block **7728** additionally internalizes the profile for well performing access (e.g. to a XML tag tree which can be processed). This alternate internalization embodiment for block **7728** would additionally require performing internalization after every time the user modified the profile, in which case there could be a special editor used by the user for creating/maintaining the profile, a special user post-edit process to cause internalization, or some other scheme for maintaining a suitable internalization. In an embodiment which internalizes the profile from a special editor, the special editor processing can also limit the user to what may be put in the profile, and validate its contents prior to internalization. An internalized profile is preferably always in correct parse-friendly form to facilitate performance when being accessed. In the embodiment of block **7728** which sets the fully qualified path name of the profile file, a special editor may still be used as described, or any suitable editor may be used, but validation and obvious error handling may have to be performed when accessing the profile, if not validated by block **7724** beyond a correct file path. Some embodiments may implement a profile in a storage embodiment that is not part of a file system.

If block **7720** determines the user did not select to enable profile information to be maintained to field **1100***k*, then processing continues to block **7730**. If block **7730** determines the user selected to exit FIG. **77** processing, application fields **1100***k* specification processing terminates at block **7732**. If block **7730** determines the user did not select to exit, then processing continues to block **7734** where any other user actions detected at block **7706** are handled appropriately. Block **7734** then continues back to block **7704**.

There can be many MS application sections of field **1100***k* which are enabled or disabled by blocks **7708** through **7714**. In the preferred embodiment of profile processing, the profile is a human readable text file, and any file of the MS can be compared to a profile of a WDR so that the user can maintain many profiles for the purpose of comparisons in expressions. Alternate embodiments include a binary file, data maintained to some storage, or any other set of data which can be processed in a similar manner as describe for profile processing. Some embodiments support specification of how to enable/disable at blocks **7708** through **7714** derivatives for mWITS, iWITS and/or oWITS.

In the preferred embodiment, a profile text file contains at least one tagged section, preferably using XML tags. Alternatively, Standard Generalized Markup Language (SGML) or HTML may be used for encoding text in the profile. There may be no standardized set of XML tags, although this would make for a universally consistent interoperability. The only requirement is that tags be used to define text strings which can be searched and compared. It helps for a plurality of users to know what tags each other uses so that comparisons can be made on a tag to tag basis between different profiles. A plurality of MS users should be aware of profile tags in use between each other so as to provide functionality for doing comparisons, otherwise profiles that use different tags cannot be compared.

Indicators disabled or enabled, as well as the profile participation variable is to be observed by WDR processing so that field **1100***k* is used accordingly. In some embodiments, certain application field sections cannot be enabled or disabled by users (i.e. a MS system setting). In preferred embodiments, WITS processing checks these settings to determine whether or not to perform applicable processing. In some embodiments, WITS processing checks these settings to strip out (e.g. for setting(s) disabled) information from a WDR which is to be in process.

FIG. **78** depicts a simplified example of a preferred XML syntactical encoding embodiment of a profile for the profile section of WDR Application Fields **1100***k*. This is also the contents of a profile file as specified at block **7724**. Any tag may have any number of subordinate tags and there can be any number of nested levels of depth of subordinate tags. A user can define his own tags. Preferably, the user anticipates what other MS users are using for tags. Individual text elements for a tag are preferably separated by semicolons. Blanks are only significant when non-adjacent to a semicolon. The text between tags is compared (e.g. text elements (e.g. Moorestown)), regardless of whether a tag contains subordinate tags, however subordinate tags are compared for matching prior to determining a match of contents between

them. Ultimately, the semicolon delimited text elements between the lowest order tags (leaf node tag sections of tag tree) are compared for matching. Ascending XML tags and the lowest level tags hierarchy provide the guide for what to compare. Thus, tags provide the map of what to compare, and the stuff being compared is the text elements between the lowest order tags of a particular tag hierarchy tree. Some explanations of atomic operator uses in expressions are described for an in-process WDR:

#d:\myprofs\benchmark.xml>5

This condition determines if the benchmark.xml file contains greater than 5 tag section matches in the entire WDR profile of the WDR in process. Text elements of the lowest order tag sections are used to decide the comparison results. A tag hierarchy, if present, facilitates how to compare. Six (six) or more matches evaluates to true, otherwise the condition evaluates to false.

% d:\myprofs\benchmarkl>=75

This condition determines if the benchmark.xml file contains greater than or equal to 75% of tag section matches in the entire WDR profile of the WDR in process. Contents that occurs between every tag is compared for a match. The number of matches found divided by the number of tag matches performed provides the percentage of matches (after multiplying the result by 100). The resulting percentage greater than or equal to 75% evaluates to true, otherwise the condition evaluates to false.

#(interests)d:\myprofs\benchmark.xml>2

In using FIG. **78** as an example, this condition determines if the benchmark.xml file contains greater than two (2) semicolon delimited matches within only the interests tag in the WDR profile of the WDR in process. If either the benchmark.xml file or the WDR profile does not contain the interests tag, then the condition evaluates to false. If both contain the interests tag, then the semicolon delimited items which is interests tag delimited are compared. Three (3) or more semicolon delimited interests that match evaluates to true, otherwise the condition evaluates to false.

% (home,hangouts)d:\myprofs\benchmark.xml>75

This condition determines if the benchmark.xml file contains greater than 75% matches when considering the two tags home and hangouts in the WDR profile of the WDR in process. Any number of tags, and any level of ascending tag hierarchy, can be specified within the (. . .) syntax. If either the benchmark.xml file or the WDR profile does not contain the tags for matching, then the condition evaluates to false. If both contain the sought tags for matching, then the text elements of the lowest order subordinate tags are treated as the items for compare. Of course, if the tags have no subordinate tags, then text elements would be compared that occurs between those tag delimiters. The number of matches found divided by the number of comparisons made provides the percentage of matches (after multiplying the result by 100). The resulting percentage greater than 75% evaluates to true, otherwise the condition evaluates to false.

WITS processing preferably uses an internalized form of FIG. **78** to perform comparisons. The internalized form may be established ahead of time as discussed above for better WITS processing performance, or may be manufactured by WITS processing in real time as needed.

Other Embodiments

As mentioned above, architecture **1900** provides a set of processes which can be started or terminated for desired functionality. Thus, architecture **1900** provides a palette from which to choose desired deployment methods for an LN expanse.

In some embodiments, all whereabouts information can be pushed to expand the LN-expanse. In such embodiments, the palette of processes to choose from includes at least process **1902**, process **1912** and process **1952**. Additionally, process **1932** would be required in anticipation of LN-expanse participating data processing systems having NTP disabled or unavailable. Additionally, process **1922** could be used for ensuring whereabouts are timely (e.g. specifically using all blocks except **2218** through **2224**). Depending on DLM capability of MSs in the LN-expanse, a further subset of processes **1902**, **1912**, **1952** and **1932** may apply. Thread(s) **1902** beacon whereabouts information, regardless of the MS being an affirmifier or pacifier.

In some embodiments, all whereabouts information can be pulled to expand the LN-expanse. In such embodiments, the palette of processes to choose from includes at least process **1922** (e.g. specifically using all blocks except **2226** and **2228**), process **1912**, process **1952** and process **1942**. Additionally, process **1932** would be required in anticipation of LN-expanse participating data processing systems having NTP disabled or unavailable. Depending on DLM capability of MSs in the LN-expanse, a further subset of processes **1922**, **1912**, **1952**, **1942** and **1932** may apply.

There are many embodiments derived from architecture **1900**. Essential components are disclosed for deployment varieties. In communications protocols which acknowledge a transmission, processes **1932** may not be required even in absence of NTP use. A sending MS appends a sent date/time stamp (e.g. field **1100***n*) on its time scale to outbound data **1302** and an acknowledging MS (or service) responds with the sent date/time stamp so that when the sending MS receives it (receives data **1302** or **1312**), the sending MS (now a receiving MS) calculates a TDOA measurement by comparing when the acknowledgement was received and when it was originally sent. Appropriate correlation outside of process **1932** deployment enables the sending MS to know which response went with which data **1302** was originally sent. A MS can make use of **19***xx* processes as is appropriate for functionality desired.

In push embodiments disclosed above, useful summary observations are made. Service(s) associated with antennas periodically broadcast (beacon) their reference whereabouts (e.g. WDR information) for being received by MSs in the vicinity. When such services are NTP enabled, the broadcasts include a sent date/time stamp (e.g. field **1100***n*). Upon receipt by a NTP enabled MS in the vicinity, the MS uses the date/time stamp of MS receipt (e.g. **1100***p*) with the date/time stamp of when sent (e.g. field **1100***n*) to calculate a TDOA measurement. Known wave spectrum velocity can translate to a distance. Upon receipt of a plurality of these types of broadcasts from different reference antennas, the MS can triangulate itself for determining its whereabouts relative known whereabouts of the reference antennas. Similarly, reference antennas are replaced by other NTP enabled MSs which similarly broadcast their whereabouts. A MS can be triangulated relative a mixture of reference antennas and other NTP enabled MSs, or all NTP enabled MSs. Stationary antenna triangulation is accomplished the same way as triangulating from other MSs. NTP use allows determining MS whereabouts using triangulation achievable in a single unidirectional broadcast of data (**1302** or **1312**). Furthermore, reference antennas (service(s)) need not communicate new data **1312**, and MSs need not communicate new data **1302**. Usual communications data **1312** are altered with a CK **1314** as

described above. Usual communications data **1302** are altered with a CK **1304** as described above. This enables a MS with not only knowing there are nearby hotspots, but also where all parties are located (including the MS). Beaconing hotspots, or other broadcasters, do not need to know who you are (the MS ID), and you do not need to know who they are in order to be located. Various bidirectional correlation embodiments can always be used for TDOA measurements.

In pull embodiments disclosed above, data processing systems wanting to determine their own whereabouts (requesters) broadcast their requests (e.g. record **2490**). Service(s) or MSs (responders) in the vicinity respond. When responders are NTP enabled, the responses include a sent date/time stamp (e.g. field **1100***n*) that by itself can be used to calculate a TDOA measurement if the requester is NTP enabled. Upon receipt by a requestor with no NTP, the requestor uses the date/time stamp of a correlated receipt (e.g. **1100***p*) with the date/time stamp of when sent (e.g. fields **1100***n* or **2450***a*) to calculate a time duration (TDOA) for whereabouts determination, as described above. New data or usual communications data applies as described above.

If NTP is available to a data processing system, it should be used whenever communicating date/time information (e.g. NTP bit of field **1100***b*, **1100***n* or **1100***p*) so that by chance a receiving data processing is also NTP enabled, a TDOA measurement can immediately be taken. In cases, where either the sending (first) data processing system or receiving (second) data processing system is not NTP enabled, then the calculating data processing system wanting a TDOA measurement will need to calculate a sent and received time in consistent time scale terms. This includes a correlated bidirectional communications data flow to properly determine duration in time terms of the calculating data processing system. In a send initiated embodiment, a first (sending) data processing system incorporates a sent date/time stamp (e.g. fields **1100***n* or **2450***a*) and determines when a correlated response is received to calculate the TDOA measurement (both times in terms of the first (sending) data processing system). In another embodiment, a second (receiving) data processing system receives a sent date/time stamp (e.g. field **1100***n*) and then becomes a first (sending) data processing as described in the send initiated embodiment. Whatever embodiment is used, it is beneficial in the LN-expanse to minimize communications traffic.

The NTP bit in date/time stamps enables optimal elegance in the LN-expanse for taking advantage of NTP when available, and using correlated transmissions when it is not. A NTP enabled MS is somewhat of a chameleon in using unidirectional data (**1302** or **1312** received) to determine whereabouts relative NTP enabled MS(s) and/or service(s), and then using bidirectional data (**1302**/**1302** or **1302**/**1312**) relative MS(s) and/or service(s) without NTP. A MS is also a chameleon when considering it may go in and out of a DLM or ILM identity/role, depending on what whereabouts technology is available at the time.

The MS ID (or pseudo MS ID) in transmissions is useful for a receiving data processing system to target a response by addressing the response back to the MS ID. Targeted transmissions target a specific MS ID (or group of MS IDs), while broadcasting is suited for reaching as many MS IDs as possible. Alternatively, just a correlation is enough to target a data source.

In some embodiments where a MS is located relative another MS, this is applicable to something as simple as locating one data processing system using the location of another data processing system. For example, the whereabouts of a cell phone (first data processing system) is used to locate an in-range automotive installed (second) data processing system for providing new locational applications to the second data processing system (or visa-versa). In fact, the second data processing may be designed for using the nearby first data processing system for determining its whereabouts. Thus, as an MS roams, in the know of its own whereabouts, the MS whereabouts is shared with nearby data processing systems for new functionality made available to those nearby data processing systems when they know their own whereabouts (by associating to the MS whereabouts). Data processing systems incapable of being located are now capable of being located, for example locating a data processing equipped shopping cart with the location of an MS, or plurality of MSs.

Architecture **1900** presents a preferred embodiment for IPC (Interprocess Communications Processing), but there are other embodiments for starting/terminating threads, signaling between processes, semaphore controls, and carrying out present disclosure processing without departing from the spirit and scope of the disclosure. In some embodiments, threads are automatically throttled up or down (e.g. **1952**-Max) per unique requirements of the MS as determined by how often threads loop back to find an entry already waiting in a queue. If thread(s) spend less time blocked on queue, they can be automatically throttled up. If thread(s) spend more time blocked on queue, they can be automatically throttled down. Timers can be associated with queue retrieval to keep track of time a thread is blocked.

LBX history **30** preferably maintains history information of key points in processing where history information may prove useful at a future time. Some of the useful points of interest may include:

- Interim snapshots of permissions **10** (for documenting who had what permissions at what time) at block **1478**;
- Interim snapshots of charters **12** (for documenting charters in effect at what times) at block **1482**;
- Interim snapshots of statistics **14** (for documenting useful statistics worthy of later browse) at block **1486**;
- Interim snapshots of service propagation data of block **1474**;
- Interim snapshots of service informant settings of block **1490**;
- Interim snapshots of LBX history maintenance/configurations of block **1494**;
- Interim snapshots of a subset of WDR queue **22** using a configured search criteria;
- Interim snapshots of a subset of Send queue **24** using a configured search criteria;
- Interim snapshots of a subset of Receive queue **26** using a configured search criteria;
- Interim snapshots of a subset of PIP data **8**;
- Interim snapshots of a subset of data **20**;
- Interim snapshots of a subset of data **36**;
- Interim snapshots of other resources **38**;
- Trace, debug, and/or dump of any execution path subset of processing flowcharts described; and/or
- Copies of data at any block of processing in any flowchart heretofore described.

Entries in LBX history **30** preferably have entry qualifying information including at least a date/time stamp of when added to history, and preferably an O/S PID and O/S TID (Thread Identifier) associated with the logged entry, and perhaps applicable applications involved (e.g. see fields **1100***k*). History **30** may also be captured in such a way there are conditions set up in advance (at block **1494**), and when those conditions are met, applicable data is captured to history **30**. Conditions can include terms that are MS system wide, and

when the conditions are met, the data for capture is copied to history. In these cases, history **30** entries preferably include the conditions which were met to copy the entry to history. Depending on what is being kept to history **30**, this can become a large amount of information. Therefore, FIG. **27** can include new blocks for pruning history **30** appropriately. In another embodiment, a separate thread of processing has a sleeper loop which when awake will prune the history **30** appropriately, either in its own processing or by invoking new FIG. **27** blocks for history **30**. A parameter passed to processing by block **2704** may include how to prune the history, including what data to prune, how old of data to prune, and any other criteria appropriate for maintaining history **30**. In fact, any pruning by FIG. **27** may include any reasonable parameters for how to prune particular data of the present disclosure.

Location applications can use the WDR queue for retrieving the most recent highest confidence entry, or can access the single instance WDR maintained (or most recent WDR of block **289** discussed above). Optimally, applications are provided with an API that hides what actually occurs in ongoing product builds, and for ensuring appropriate semaphore access to multi-threaded accessed data.

Correlation processing does not have to cause a WDR returned. There are embodiments for minimal exchanges of correlated sent date/time stamps and/or received date/time stamps so that exchanges are very efficient using small data exchanges. Correlation of this disclosure was provided to show at least one solution, with keeping in mind that there are many embodiments to accomplish relating time scales between data processing systems.

Architecture **1900** provides not only the foundation for keeping an MS abreast of its whereabouts, but also the foundation upon which to build LBX nearby functionality. Whereabouts of MSs in the vicinity are maintained to queue **22**. Permissions **10** and charters **12** can be used for governing which MSs to maintain to queue **22**, how to maintain them, and what processing should be performed. For example, MS user Joe wants to alert MS user Sandy when he is in her vicinity, or user Sandy wants to be alerted when Joe is in her vicinity. Joe configures permissions enabling Sandy to be alerted with him being nearby, or Sandy configured permissions for being alerted. Sandy accepts the configuration Joe made, or Joe accepts the configuration Sandy made. Sandy's queue **22** processing will ensure Joe's WDRs are processed uniquely for desired functionality.

FIG. **8**C was presented in the context of a DLM, however architecture **1900** should be applied for enabling a user to manually request to be located with ILM processing if necessary. Blocks **862** through **870** are easily modified to accomplish a WDR request (like blocks **2218** through **2224**). In keeping with current block descriptions, block **872** would become a new series of blocks for handling the case when DLM functionality was unsuccessful. New block **872**-A would broadcast a WDR request soliciting response (see blocks **2218** through **2224**). Thereafter, a block **872**-B would wait for a brief time, and subsequently a block **872**-C would check if whereabouts have been determined (e.g. check queue **22**). Thereafter, if a block **872**-D determines whereabouts were not determined, an error could be provided to the user, otherwise the MS whereabouts were successfully determined and processing continues to block **874**. Applications that may need whereabouts can now be used. There are certainly emergency situations where a user may need to rely on other MSs in the vicinity for being located. In another embodiment, LBX history can be accessed to at least provide a most recent location, or most recently traveled set of locations, hopefully providing enough information for reasonably locating the user in the event of an emergency, when a current location cannot be determined.

To maintain modularity in interfaces to queues **24** and **26**, parameters may be passed rather than having the modular send/receive processing access fields of application records. When WDRs are "sent", the WDR will be targeted (e.g. field **1100***a*), perhaps also with field **1100***f* indicating which communications interface to send on (e.g. MS has plurality of comm. interfaces **70**). When WDRs are "broadcast" (e.g. null MS ID), the WDR is preferably outbound on all available comm. interfaces **70**), unless field **1100***f* indicates to target a comm. interface. Analogously, when WDR requests are "sent", the request will be targeted (e.g. field **2490***a*), perhaps also with field **2490***d* indicating which communications interface to send on (e.g. MS has plurality of comm. interfaces **70**). When WDR requests are "broadcast" (e.g. null MS ID), the WDR is preferably outbound on all available comm. interfaces **70**), unless field **1100***f* indicates to target a comm. interface.

Fields **1100***m*, **1100***n*, **1100***p*, **2490***b* and **2490***c* are also of interest to the transport layer. Any subset, or all, of transport related fields may be passed as parameters to send processing, or received as parameters from receiving processing to ensure send and receive processing is adaptable using pluggable transmission/reception technologies.

An alternate embodiment to the BESTWDR WDR returned by FIG. **26**B processing may be set with useful data for reuse toward a future FIG. **26**B processing thread whereabouts determination. Field **1100***f* (see pg. 168) can be set with useful data for that WDR to be in turn used at a subsequent whereabouts determination of FIG. **26**B. This is referred to as Recursive Whereabouts Determination (RWD) wherein ILMs determine WDRs for their whereabouts and use them again for calculating future whereabouts (by populating useful TDOA, AOA, MPT and/or whereabouts information to field **1100***f*).

An alternate embodiment may store remote MS movement tolerances (if they use one) to WDR field **1100***f* so the receiving MS can determine how stale are other WDRs in queue **22** from the same MS, for example when gathering all useful WDRs to start with in determining whereabouts of FIG. **26**B processing (e.g. block **2634**). Having movement tolerances in effect may prove useful for maximizing useful WDRs used in determining a whereabouts (FIG. **26**B processing).

Many LBX aspects have been disclosed, some of which are novel and new in LBS embodiments. While it is recommended that features disclosed herein be implemented in the context of LBX, it may be apparent to those skilled in the art how to incorporate features which are also new and novel in a LBS model, for example by consolidating distributed permission, charters, and associated functionality to a shared service connected database.

Privileges and/or charters may be stored in a datastream format (e.g. X.409), syntactical format (e.g. XML, source code (like FIGS. **51**A and **51**B), compiled or linked programming data, database data (e.g. SQL tables), or any other suitable format. Privileges and/or charters may be communicated between MSs in a datastream format (e.g. X.409), syntactical format (e.g. XML, source code (like FIGS. **51**A and **51**B), compiled or linked programming data, database data (e.g. SQL tables), or any other suitable format.

Block **4466** may access an all or none permission (privilege) to receive permission and/or charter data (depending on what data is being received) from a particular identity (e.g. user or particular MS). Alternate embodiments implement more granulated permissions (privileges) on which types,

sets, or individual privileges and/or charters can be received so that block **4470** will update local data with only those privileges or charters that are permitted out of all data received. One embodiment is to receive all privileges and/or charters from remote systems for local maintaining so that FIG. **57** processing can later determine what privileges and charters are enabled. This has the benefit for the receiving user to know locally what the remote user(s) desire for privileges and charters without them necessarily being effective. Another embodiment is for FIG. **44**B to only receive the privileged subset of data that can be used (privileged) at the time, and to check at block **4466** which privileges should be used to alter existing privileges or charters from the same MS (e.g. altered at block **4470**). This has the potential benefit of less MS data to maintain and better performance in FIG. **57** processing for dealing only with those privileges and charters which may be useable. A user may still browse another user's configurations with remote data access anyway.

WPL is a unique programming language wherein peer to peer interaction events containing whereabouts information (WDRs) provide the triggers for novel location based processing, however a LBS embodiment may also be pursued. Events seen, or collected, by a service may incorporate WPL, the table record embodiments of FIGS. **35**A through **37**C, a suitable programming executable and/or data structures, or any other BNF grammar derivative to carry out analogous event based processing. For example, the service would receive inbound whereabouts information (e.g. WDRS) from participating MSs and then process accordingly. An inbound, outbound, and in-process methodology may be incorporated analogously by processing whereabouts information from MSs as it arrives to the service (inbound), processing whereabouts information as it is sent out from the service (outbound) to MSs, and processing whereabouts information as it is being processed by the service (in process) for MSs. In one embodiment, service informant code **28** is used to keep the service informed of the LBX network. In another embodiment, a conventional LBS architecture is deployed for collecting whereabouts of MSs.

An alternate embodiment processes inbound/outbound/maintained WDRs in process transmitted to a MS from non-mobile data processing systems, perhaps data processing systems which are to emulate a MS, or perhaps data processing systems which are to contribute to LBX processing. Interoperability is as disclosed except data processing systems other than MSs participate in interacting with WDRs. In other embodiments, the data processing systems contain processing disclosed for MSs to process WDRs from MSs (e.g. all disclosed processing or any subset of processing (e.g. WITS processing)).

Communications between MSs and other MSs, or between MSs and data processing systems, may be compressed, encrypted, and/or encoded for performance or concealing. Any protocol, X.409 encodings, datastream encodings, or other data which is critical for processing shall have integrity regardless of an encapsulating or embedded encoding that may be in use. Further, internalizations of the BNF grammar may also be compressed, encrypted, and/or encoded for performance or concealing. Regardless of an encapsulating or embedded encoding that may be in use, integrity shall be maintained for processing. When other encodings are used (compression, encryption, etc), an appropriate encode and decode pair of processing is used (compress/decompress, encrypt/decrypt, etc).

Grammar specification privileges are preferably enforced in real time when processing charters during WITS processing. For example, charters specified may initially be ineffective, but can be subsequently enabled with a privilege. It is preferred that privileges **10** and charters **12** be maintained independently during configuration time, and through appropriate internalization. This allows specifying anything a user wants for charters, regardless of privileges in effect at the time of charter configuration, so as to build those charters which are desired for processing, but not necessarily effective yet. Privileges can then be used to enable or disable those charters as required. In an alternate embodiment, privileges can be used to prevent certain charters from even being created. This helps provide an error to the user at an appropriate time (creating an invalid charter), however a valid charter may lose a privilege later anyway and become invalid. The problem of a valid charter becoming invalid later has to be dealt with anyway (rather than automatically deleting the newly invalid charter). Thus, it is preferable to allow any charters and privileges to be specified, and then candidate for interpreting at WITS processing time.

Many embodiments are better described by redefining the "W" in acronyms used throughout this disclosure for the more generic "Wireless" use, rather than "Whereabouts" use. Thus, WDR takes on the definition of Wireless Data Record. In various embodiments, locational information fields become less relevant, and in some embodiments mobile location information is not used at all. As stated above with FIG. **11**A, when a WDR is referenced in this disclosure, it is referenced in a general sense so that the contextually reasonable subset of the WDR of FIG. **11**A is used. This notion is taken steps further.

A WDR **1100** may be redefined with a core section containing only the MS ID field **1100***a*. The MS ID field **1100***a* facilitates routing of the WDR, and addressing a WDR, for example in a completely wireless transmission of FIGS. **13**A through **13**C. In an embodiment with a minimal set of WDR fields, the WDR may contain only two (2) fields: a MS ID field **1100***a* and application fields **1100***k*. In an embodiment with minimal changes to the architecture heretofore disclosed, all WDR **1100** fields **1100***b* through **1100***p* are maintained to field **1100***k*. Disclosure up to this point continues to incorporate processing heretofore described, except WDR fields which were peers to application fields **1100***k* in a WDR **1100** are now subordinate to field **1100***k*. However, the field data is still processed the same way as disclosed, albeit with data being maintained subordinate to field **1100***k*. Thus, field **1100***k* may have broader scope for carrying the data, or for carrying similar data.

In a more extreme embodiment, a WDR (Wireless Data Record) will contain only two fields: a MS ID field **1100***a* and application fields **1100***k*; wherein a single application (or certain applications) of data is maintained to field **1100***k*. For example, the WDR is emitted from mobile MSs as a beacon which may or may not be useful to receiving MSs, however the beaconed data is for one application (other embodiments can be for a plurality of applications). In this minimal embodiment, a minimal embodiment of architecture **1900** is deployed with block changes removing whereabouts/location processing. The following processes may provide such a minimal embodiment palette for implementation:

Wireless Broadcast Thread(s) **1902**—

FIG. **20** block **2010** would be modified to "Peek WDR queue for most recent WDR with MS ID=this MS". Means would be provided for date/time stamps maintained to queue **22** for differentiating between a plurality of WDRs maintained so the more recent can be retrieved. This date/time stamp may or may not be present in a WDR during transmission which originated from a remote MS (i.e. in the WDR transmitted (beaconed)). Regardless, a date/time stamp is

preferably maintained in the WDR of queue **22**. Appropriate and timely queue **22** pruning would be performed for one or more relevant WDRs at queue **22**. FIG. **20** would broadcast at least the MS ID field **1100***a* and application data field **1100***k* for the application.

Wireless Collection Thread(s) **1912**—

FIG. **21** would be modified to remove location determination logic and would collect WDRs received that are relevant for the receiving MS and deposit them to queue **22**, preferably with a date/time stamp. Relevance can be determined by if there are permissions or charters in place for the originating MS ID at the receiving MS (i.e. WITS filtering and processing). The local MS applicable could access WDRs from queue **22** as it sees fit for processing in accordance with the application, as well as privileges and charters.

Wireless Supervisor Thread(s) **1922**—

FIG. **22** block **2212** would be modified to "Peek WDR queue for MS ID=this MS, and having a reasonably current date/time stamp" to ensure there is at least one timely WDR contained at queue **22** for this MS. If there is not a timely WDR at the MS, then processing of block **2218** through **2228** would be modified to request helpful WDRs from MSs within the vicinity, assuming the application applicable warrants requesting such help, otherwise blocks **2218** through **2228** would be modified to trigger local MS processing for ensuring a timely WDR is deposited to queue **22**.

Wireless Data Record Request Thread(s) **1942**—

FIG. **25** block **2510** would be modified to "Peek WDR queue for most recent WDR with this MS ID" and then sending/broadcasting the response to the requesting MS. FIG. **25** would be relevant in an architecture wherein the application does in fact rely on MSs within the vicinity for determining its own WDRs.

One application using such a minimal embodiment may be the transmission of profile information (see # and % operators above). As a MS roams, it beacons out its profile information for other MSs to receive it. The receiving MSs then decide to process the profile data in fields **1100***k* according to privileges and/or charters that are in place. Note that there is no locating information of interest. Only the profile information is of interest. Thus, the MSs become wireless beacons of data that may or may not be processed by receiving MSs within the wireless vicinity of the originating MS. Consider a singles/dating application wherein the profile data contains characteristics and interests of the MS user. A privilege or charter at the receiving MS could then process the profile data when it is received, assuming the receiving MS user clarified what is of interest for automated processing through configurations for WITS processing.

While a completely wireless embodiment is the preferred embodiment since MS users may be nearby by virtue of a completely wireless transmission, a longer range transmission could be facilitated by architectures of FIGS. **50**A through **50**C. In an architecture of transmission which is not completely wireless, the minimal embodiment WDR would include field(s) indicating a route which was not completely wireless, perhaps how many hops, etc as disclosed above. WITS filtering would play an important role to ensure no outbound transmissions occur unless there are configurations in place that indicate a receiving MS may process it (i.e. there are privileges and/or charters in place), and no inbound processing occurs unless there are appropriate configurations in place for the originating MS(s) (i.e. there are privileges and/or charters in place). Group identities of WDRs can become more important as a criteria for WITS filtering, in particular when a group id indicates the type of WDR. The longer range embodiment of FIG. **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.

While various embodiments of the present disclosure have been described above, it should be understood that they have been presented by way of example only, and not limitation. Thus, the breadth and scope of the present disclosure should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

What is claimed is:

1. A method for automatic location based exchange processing by a mobile data processing system, the method comprising:

presenting a user interface to a user of the mobile data processing system, the user interface for configuring privilege data relating the mobile data processing system with a remote data processing system, the privilege data stored local to the mobile data processing system and searched upon receipt of whereabouts data received for processing by the mobile data processing system;

receiving, for processing by the mobile data processing system, the whereabouts data including an originating identity of the whereabouts data;

searching, by the mobile data processing system, the privilege data stored local to the mobile data processing system for a matching privilege upon the receiving, for processing by the mobile data processing system, the whereabouts data, wherein the matching privilege is configured for relating the originating identity of the whereabouts data with a destination identity of the whereabouts data to permit trigger of a privileged action for the receipt of whereabouts data received for processing by the mobile data processing system; and

performing the privileged action at the mobile data processing system upon finding the matching privilege, after the searching, by the mobile data processing system, the privilege data stored local to the mobile data processing system.

2. The method of claim **1** wherein the privileged action is configured by a user of the remote data processing system.

3. The method of claim **2** wherein the destination identity is associated to the mobile data processing system and wherein the receiving, for processing by the mobile data processing system, the whereabouts data including an originating identity of the whereabouts data comprises receiving, for processing by the mobile data processing system, inbound whereabouts data including an originating identity of the whereabouts data, wherein the originating identity is associated to the remote data processing system, and wherein the whereabouts data is sent by the remote data processing system.

4. The method of claim **2** wherein the destination identity is associated to the remote data processing system and wherein the receiving, for processing by the mobile data processing system, the whereabouts data including an originating identity of the whereabouts data comprises receiving, for processing by the mobile data processing system, outbound whereabouts data including an originating identity of the whereabouts data, wherein the originating identity is associated to the mobile data processing system, and wherein the whereabouts data is to be sent to the remote data processing system.

5. The method of claim **1** wherein the privileged action is configured by the user of the mobile data processing system.

6. The method of claim 5 wherein the destination identity is associated to the mobile data processing system and wherein the receiving, for processing by the mobile data processing system, the whereabouts data including an originating identity of the whereabouts data comprises receiving, for processing by the mobile data processing system, inbound whereabouts data including an originating identity of the whereabouts data, wherein the originating identity is associated to the remote data processing system, and wherein the whereabouts data is sent by the remote data processing system.

7. The method of claim 5 wherein the destination identity is associated to the remote data processing system and wherein the receiving, for processing by the mobile data processing system, the whereabouts data including an originating identity of the whereabouts data comprises receiving, for processing by the mobile data processing system, outbound whereabouts data including an originating identity of the whereabouts data, wherein the originating identity is associated to the mobile data processing system, and wherein the whereabouts data is to be sent to the remote data processing system.

8. The method of claim 1 further including:

maintaining a user configured charter at the mobile data processing system, the charter having a conditional expression and an associated action depending on evaluation of the conditional expression;

evaluating the conditional expression by comparing the conditional expression to the whereabouts data, upon the receiving, for processing by the mobile data processing system, the whereabouts data; and

performing the associated action at the mobile data processing system upon the evaluating the conditional expression by comparing the conditional expression to the whereabouts data.

9. The method of claim 8 wherein the charter is configured by a user of the remote data processing system.

10. The method of claim 8 wherein the maintaining a user configured charter at the mobile data processing system comprises maintaining a user specified textual syntax.

11. The method of claim 10 wherein the user specified textual syntax comprises an XML encoding.

12. The method of claim 10 wherein the user specified textual syntax comprises a Whereabouts Programming Language encoding.

13. The method of claim 1 wherein the whereabouts data is carried by way of a wireless communications transmission through no intervening data processing system between the mobile data processing system and the remote data processing system.

14. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with a user configured time specification, the time specification stored local to the mobile data processing system and used to compare to a receipt time of the receipt of whereabouts data received for processing by the mobile data processing system.

15. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes initiating an action at the remote data processing system.

16. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes sending an sms message.

17. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes sending an electronic mail.

18. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes automatically making a phone call by the mobile data processing system.

19. The method of claim 15 wherein the remote data processing system establishes a phone call with the mobile data processing system.

20. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes presenting information to an informative user interface.

21. The method of claim 1 wherein the whereabouts data is an unsolicited broadcast of data from the remote data processing system.

22. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, a specified distance between locations of the mobile data processing system and the remote data processing system.

23. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, the remote data processing system is at a specified location.

24. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, the remote data processing system is at a specified situational location.

25. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, the remote data processing system arrived to a specified location during a time in history.

26. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, the remote data processing system departed a specified location during a time in history.

27. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, the mobile data processing system is in a specified vicinity of a plurality of other mobile data processing systems.

28. The method of claim 1 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes altering calendar application data.

29. A mobile data processing system comprising:

one or more processors; and

memory coupled to the one or more processors and storing instructions, which when executed by the one or more processors, causes the one or more processors to perform operations comprising:

presenting a user interface to a user of the mobile data processing system, the user interface for configuring privilege data relating the mobile data processing system with a remote data processing system, the privilege data stored local to the mobile data processing system and searched upon receipt of whereabouts data received for processing by the mobile data processing system;

receiving, for processing by the mobile data processing system, the whereabouts data including an originating identity of the whereabouts data;

searching, by the mobile data processing system, the privilege data stored local to the mobile data processing system for a matching privilege upon the receiving, for processing by the mobile data processing system, the whereabouts data, wherein the matching privilege is configured for relating the originating identity of the whereabouts data with a destination identity of the whereabouts data to permit trigger of a privileged action for the receipt of whereabouts data received for processing by the mobile data processing system; and

performing the privileged action at the mobile data processing system upon finding the matching privilege, after the searching, by the mobile data processing system, the privilege data stored local to the mobile data processing system.

30. The system of claim **29** wherein the whereabouts data is an unsolicited broadcast of data from the remote data processing system.

31. The system of claim **29** wherein the privileged action is configured by a user of the remote data processing system.

32. The system of claim **31** wherein the destination identity is associated to the mobile data processing system and wherein the receiving, for processing by the mobile data processing system, the whereabouts data including an originating identity of the whereabouts data comprises receiving, for processing by the mobile data processing system, inbound whereabouts data including an originating identity of the whereabouts data, wherein the originating identity is associated to the remote data processing system, and wherein the whereabouts data is sent by the remote data processing system.

33. The system of claim **31** wherein the destination identity is associated to the remote data processing system and wherein the receiving, for processing by the mobile data processing system, the whereabouts data including an originating identity of the whereabouts data comprises receiving, for processing by the mobile data processing system, outbound whereabouts data including an originating identity of the whereabouts data, wherein the originating identity is associated to the mobile data processing system, and wherein the whereabouts data is to be sent to the remote data processing system.

34. The system of claim **29** wherein the privileged action is configured by the user of the mobile data processing system.

35. The system of claim **34** wherein the destination identity is associated to the mobile data processing system and wherein the receiving, for processing by the mobile data processing system, the whereabouts data including an originating identity of the whereabouts data comprises receiving, for processing by the mobile data processing system, inbound whereabouts data including an originating identity of the whereabouts data, wherein the originating identity is associated to the remote data processing system, and wherein the whereabouts data is sent by the remote data processing system.

36. The system of claim **34** wherein the destination identity is associated to the remote data processing system and wherein the receiving, for processing by the mobile data processing system, the whereabouts data including an originating identity of the whereabouts data comprises receiving, for processing by the mobile data processing system, outbound whereabouts data including an originating identity of the whereabouts data, wherein the originating identity is associated to the mobile data processing system, and wherein the whereabouts data is to be sent to the remote data processing system.

37. The system of claim **29** wherein the operations further include:

maintaining a user configured charter at the mobile data processing system, the charter having a conditional expression and an associated action depending on evaluation of the conditional expression;

evaluating the conditional expression by comparing the conditional expression to the whereabouts data, upon the receiving, for processing by the mobile data processing system, the whereabouts data; and

performing the associated action at the mobile data processing system upon the evaluating the conditional expression by comparing the conditional expression to the whereabouts data.

38. The system of claim **37** wherein the charter is configured by a user of the remote data processing system.

39. The system of claim **37** wherein the maintaining a user configured charter at the mobile data processing system comprises maintaining a user specified textual syntax.

40. The system of claim **39** wherein the user specified textual syntax comprises an XML encoding.

41. The system of claim **39** wherein the user specified textual syntax comprises a Whereabouts Programming Language encoding.

42. The system of claim **29** wherein the whereabouts data is carried by way of a wireless communications transmission through no intervening data processing system between the mobile data processing system and the remote data processing system.

43. The system of claim **29** wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with a user configured time specification, the time specification stored local to the mobile data processing system and used to compare to a receipt time of the receipt of whereabouts data received for processing by the mobile data processing system.

44. The system of claim **29** wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes initiating an action at the remote data processing system.

45. The system of claim **44** wherein the remote data processing system establishes a phone call with the mobile data processing system.

46. The system of claim **29** wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes sending an sms message.

47. The system of claim **29** wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes sending an electronic mail.

48. The system of claim **29** wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes automatically making a phone call by the mobile data processing system.

49. The system of claim 29 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes presenting information to an informative user interface.

50. The system of claim 29 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, a specified distance between locations of the mobile data processing system and the remote data processing system.

51. The system of claim 29 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, the remote data processing system is at a specified location.

52. The system of claim 29 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, the remote data processing system is at a specified situational location.

53. The system of claim 29 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, the remote data processing system arrived to a specified location during a time in history.

54. The system of claim 29 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, the remote data processing system departed a specified location during a time in history.

55. The system of claim 29 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes performing the privileged action in accordance with determining, by the mobile data processing system, the mobile data processing system is in a specified vicinity of a plurality of other mobile data processing systems.

56. The system of claim 29 wherein the performing the privileged action at the mobile data processing system upon finding the matching privilege includes altering calendar application data.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

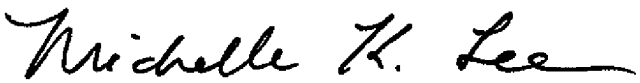
PATENT NO. : 8,639,267 B2
APPLICATION NO. : 12/287064
DATED : January 28, 2014
INVENTOR(S) : William J. Johnson

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In the Claims:

In Col. 287, line 28 (Claim 30), please insert -- mobile data processing -- before the word "system".
In Col. 287, line 31 (Claim 31), please insert -- mobile data processing -- before the word "system".
In Col. 287, line 33 (Claim 32), please insert -- mobile data processing -- before the word "system".
In Col. 287, line 44 (Claim 33), please insert -- mobile data processing -- before the word "system".
In Col. 287, line 55 (Claim 34), please insert -- mobile data processing -- before the word "system".
In Col. 287, line 57 (Claim 35), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 1 (Claim 36), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 12 (Claim 37), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 26 (Claim 38), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 28 (Claim 39), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 31 (Claim 40), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 33 (Claim 41), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 36 (Claim 42), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 41 (Claim 43), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 49 (Claim 44), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 53 (Claim 45), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 56 (Claim 46), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 60 (Claim 47), please insert -- mobile data processing -- before the word "system".
In Col. 288, line 64 (Claim 48), please insert -- mobile data processing -- before the word "system".
In Col. 289, line 1 (Claim 49), please insert -- mobile data processing -- before the word "system".
In Col. 289, line 5 (Claim 50), please insert -- mobile data processing -- before the word "system".
In Col. 289, line 12 (Claim 51), please insert -- mobile data processing -- before the word "system".
In Col. 289, line 18 (Claim 52), please insert -- mobile data processing -- before the word "system".
In Col. 290, line 1 (Claim 53), please insert -- mobile data processing -- before the word "system".
In Col. 290, line 7 (Claim 54), please insert -- mobile data processing -- before the word "system".
In Col. 290, line 13 (Claim 55), please insert -- mobile data processing -- before the word "system".
In Col. 290, line 20 (Claim 56), please insert -- mobile data processing -- before the word "system".

Signed and Sealed this
Sixth Day of May, 2014



Michelle K. Lee
Deputy Director of the United States Patent and Trademark Office