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

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

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

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

If block **4734** determines the user selected to modify an action, then block **4736** interfaces with the user to modify action data of the entry pointed to by the list cursor. The user may change information of the ADR and any associated records (e.g. DDR, TDR). The user may also add the associated records at block **4736**. Block **4736** waits for a user action indicating completion. Block **4736** will continue to block **4738** when the action is detected at block **4736**. If block **4738** determines the user exited, then processing continues back to block **4712**. If block **4738** determines the user selected to save changes made at block **4736**, then block **4740** updates the data and the list is appropriately updated before continuing back to block **4712**. Block **4740** may update the ADR and/or any associated records (e.g. DDR and/or TDR) using the action id field **3750***a* (associated to the action item at block **4710**). Block **4740** will update an associated HDR as well. Block **4736** may add a new a DDR and/or TDR as part of the action change. If block **4734** determines the user did not select to modify an action, then processing continues to block **4752** by way of off-page connector **4750**.

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

If block **4758** determines the user selected to delete an action, then block **4760** determines any data records (e.g. CDR(s)) that reference the action data record to be deleted. Preferably, no referencing data records (e.g. CDRs) are joinable (e.g. field **3700***d*) to the action data record being deleted, otherwise the user may improperly delete an action from a configured charter. The user should remove ascending references to an action for deletion first. Block **4760** continues to block **4762**. If block **4762** determines there was at least one

CDR reference, block **4764** provides an appropriate error with the reference(s) found so the user can subsequently reconcile. Block **4764** preferably requires the user to acknowledge the error before continuing back to block **4712**. If no references were found as determined by block **4762**, then processing continues to block **4766** for deleting the data record currently pointed to by the list cursor. Block **4766** also modifies the list for the discarded entry, and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **4712**. Block **4766** will use the action ID field **3750***a* (associated with the entry at block **4710**) to delete an action. Associated records (e.g. DDR **3600**, HDR **3620**, and TDR **3640**) are also deleted (e.g. preferably with a cascade delete in a SQL embodiment). If block **4758** determines the user did not select to delete an action, then processing continues to block **4768**.

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

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

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

If block **4818** determines the user selected to set the list cursor to a different parameter entry, then block **4820** sets the list cursor accordingly and processing continues back to block **4812**. Block **4812** always sets for indicating where the list cursor is currently pointed and sets for appropriately scrolling the list if necessary when subsequently presenting the list at block **4814**. If block **4818** determines the user did not select to set the list cursor, then processing continues to block **4822**. If block **4822** determines the user selected to add a parameter, then block **4824** accesses a maximum number of parameter entries allowed (perhaps multiple maximum values accessed), and block **4826** checks the maximum(s) with the number of current parameter entries defined. There are many embodiments for what deems a maximum (for this user, for a group, for this MS, etc). If block **4826** determines a maximum number of parameter entries allowed already exists, then block **4828** provides an error to the user and processing continues back to block **4812**. Block **4828** preferably requires the user to acknowledge the error before continuing back to block **4812**. If block **4826** determines a maximum was not exceeded, then block **4830** interfaces with the user for entering validated parameter data, and block **4832** adds the data record, appropriately updates the list with the new entry, and sets the list cursor appropriately for the next list presentation refresh, before continuing back to block **4812**. If block **4822** determines the user did not want to add a parameter entry, processing continues to block **4834**. Block **4832** will add a PARMDR, DDR **3600** and HDR **3620** (to set creator information). The DDR is optionally added by the user.

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

With reference now to FIG. **48**B, if block **4852** determines the user selected to get more details of the parameter entry, then block **4854** gets additional details (may involve database queries in an SQL embodiment) for the parameter entry

pointed to by the list cursor, and block **4856** appropriately presents the information to the user. Block **4856** then waits for a user action that the user is complete reviewing details, in which case processing continues back to block **4812** by way of off-page connector **4898**. If block **4852** determines the user did not select to get more detail, then processing continues to block **4858**.

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

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

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

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

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

1) The first identity ID_1 (Grantor) granting a privilege to a second identity ID_2 (Grantee; grammar ID/IDType), as shown in cell **4924**: Privilege data is maintained by ID_1 at the ID_1 MS as is used to govern actions, functionality, features, and/or behavior for the benefit of ID_2, by a) processing ID_1 WDR information at the ID_2 MS (preferably, privileges are communicated to ID_2 MS for enforcing and/or cloning there), b) processing ID_2 WDR information at the ID_1 MS (privileges locally maintained to ID_1), and c) processing ID_1 WDR information at the ID_1 MS (privileges locally maintained to ID_1);
2) The first identity ID_1 (Grantor) granting a privilege to himself (Grantee), as shown in cell **4922**: Preferably, privilege data in this case is not necessary, no configuration interface is required for this scenario, and an identity implicitly has all conceivable privileges assigned to himself by default; however, alternatively privileges may be appropriate for activating/deactivating functionality;
3) The second identity ID_2 (Grantor) granting a privilege to the first identity (Grantee), as shown in cell **4926**: Privilege data is used for informing ID_1 (or enabling ID_1 to clone per a privilege) and to govern actions, functionality, features, and/or behavior for the benefit of ID_1, by a) processing ID_2 WDR information at the ID_1 MS (preferably, privileges are communicated to ID_1 MS for enforcing and/or cloning there), b) processing ID_1 WDR information at the ID_2 MS (privileges locally maintained to ID_2); and c) processing ID_2 WDR information at the ID_2 MS (privileges locally maintained to ID_2); and/or
4) The second identity granting a privilege to himself, as shown in cell **4928**: Preferably, privilege data in this case is not necessary, no communications interface is required for this scenario, and an identity implicitly has all conceivable privileges assigned to himself by default; however, alternatively privileges may be appropriate for activating/deactivating functionality.

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

5) A first identity ID_1 (Grantor) granting a privilege to the second identity ID_2 (Grantee; grammar ID/IDType), as shown in cell **4944**: Privilege data is used for informing

ID_2 (or enabling ID_2 to clone per a privilege) and to govern actions, functionality, features, and/or behavior for the benefit of ID_2, by a) processing ID_1 WDR information at the ID_2 MS (preferably, privileges are communicated to ID_1 MS for enforcing and/or cloning there), b) processing ID_2 WDR information at the ID_1 MS (privileges locally maintained to ID_1), and c) processing ID_1 WDR information at the ID_1 MS (privileges locally maintained to ID_1);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

queue **22**); and b) incoming to a particular MS (e.g. "incoming" as being communicated to the MS).

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

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

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

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

Data processing system **5000** may be a DLM, ILM, or service being communicated with by DML **200***a* as disclosed in the present disclosure for FIGS. **13**A through **13**C, or for FIGS. **50**A through **50**C. LBX architecture is founded on peer to peer interaction between MSs without requiring a service to middleman data, however data processing systems **5090**, **5092** and those applicable to connection **5094** can facilitate the peer to peer interactions. In some embodiments, data processing systems between DLM **200***a* and the data processing **5000** intercept data for tracking, book-keeping, statistics, and for maintaining data potentially accessed by service informant code **28**, however, the LBX peer to peer model is preferably not interfered with. Data processing system **5000** generically represents a DLM, ILM or service(s) for analogous FIGS. **13**A through **13**C processing for sending/broadcasting data such as a data packet **5002** (like **1302**/**1312**). When a Communications Key (CK) **5004** (like **1304**/**1314**) is embedded within data **5002**, data **5002** is considered usual communications data (e.g. protocol, voice, or any other data over conventional forward channel, reverse channel, voice data channel, data transmission channel, or any other appropriate channel) which has been altered to contain CK **5004**. Data **5002** contains a CK **5004** which can be detected, parsed, and processed when received by an MS or other data processing system in the vicinity (conceivably any distance depending on embodiment) of data processing system **5000** as determined by the maximum range of transmission **5006** (like **1306**/**1316**). CK **5004** permits "piggy-backing" on current transmissions to accomplish new functionality as disclosed herein. Transmissions radiate out in all directions in a manner consistent with the wave spectrum used, and data carried thereon may or may not be encrypted (e.g. encrypted WDR information). The radius **5008** (like **1308**/**1318**) represents a first range of signal reception from data processing system **5000** (e.g. antenna thereof), perhaps by a MS. The radius **5010** (like **1310**/**1320**) represents a second range of signal reception from data processing system **5000** (e.g. antenna thereof), perhaps by a MS. The radius **5011** (like **1311**/**1322**) represents a third range of signal reception from data processing system **5000** (e.g. antenna thereof), perhaps by a MS. The radius **5006** (like **1306**/**1316**) represents a last and maximum range of signal reception from data processing system **5000** (e.g. antenna thereof), perhaps by a MS (not shown). The time of transmission from data processing system **5000** to radius **5008** is less than times of transmission from service to radi-

uses **5010**, **5011**, or **5006**. The time of transmission from data processing system **5000** to radius **5010** is less than times of transmission to radiuses **5011** or **5006**. The time of transmission from data processing system **5000** to radius **5011** is less than time of transmission to radius **5006**. In another embodiment, data **5002** contains a Communications Key (CK) **5004** because data **5002** is new transmitted data in accordance with the present disclosure. Data **5002** purpose is for carrying CK **5004** information for being detected, parsed, and processed when received by another MS or data processing system in the vicinity (conceivably any distance depending on embodiment) of data processing system **5000** as determined by the maximum range of transmission.

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

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

In an LN-expanse, it is important to know whether or not WDR information is of value for locating the receiving MS, for example to grow an LN-expanse with newly located MSs. FIGS. **50**A through **50**C demonstrate that WDR information sources may be great distances (over a variety of communications paths) from a particular MS receiving the WDR information. Carrying intermediary system indication is well known in the art, for example to know the number of hops of a communications path. The preferred embodiment uses communications reference field **1100***g* to maintain whether or not the WDR encountered any intermediate systems, for example as identified with hops, network address change(s), channel extender transmission indications, or any pertinent data to indicate whether the WDR encountered anything other than a wireless transmission (e.g. directly between the sending MS and receiving MS). This provides FIG. **26**B with a means to qualify the peek at block **2634** for only those WDRs which show field **1100***g* to be over a single wireless connection from the source to the MS (i.e. block **2634** to read as "Peek all WDRS from queue **22** for confidence>confidence floor and most recent in trailing f(WTV) period of time and field **1100***g* indicating a wireless connected source over no intermediary systems"). Field **1100***g* would be set intelligently for all WDRs received and processed by the MS (e.g. inserted to queue **22**). In another embodiment, fields **1100***e* and **1100***f* are used to indicate that the WDR can be relied upon for triangulating a new location of the MS (e.g. block **2660** altered to get the next WDR from the REMOTE_MS list which did not arrive except through a single wireless path). In other embodiments, the correlation (e.g. field **1100***m*) can be used to know whether it involved more than a single wireless communications path. The requirement is to be able to distinguish between WDRs that can contribute to locating a MS

and WDRs which should not be used to locate the MS. In any case, WDRs are always useful for peer to peer interactions as governed by privileges and charters (see WITS filtering discussed below).

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

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

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

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

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

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

Block **1496***a* checks to see if the user selected to configure the WRC—an option for configuration at block **1406** wherein the user action to configure it is detected at block **1408**;

Block **1496***b* is processed if block **1496***a* determines the user did select to configure the WRC. Block **1496***b* interfaces with the user for a WRC setting (e.g. a block **1496***b*-**1** to prepare parameters for FIG. **18** processing, and a block **1496***b*-**2** for invoking the Configure value procedure of FIG. **18** to set the WRC). Processing then continues to block **1496***c*.

Block **1496***c* is processed if block **1496***a* determines the user did not select to configure the WRC, or as the result of processing leaving block **1496***b*. Block **1496***c* handles other user interface actions leaving block **1408** (e.g. becomes the "catch all" as currently shown in block **1496** of FIG. **14**B).

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

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

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

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

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

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

The assignPrivs variable is of type Generic and is set with a long string containing lots of stuff. Generic tells the internalizer to treat the assigned value as text string without any

variable type validation at this time. The BNF grammar showed that variables have a type to facilitate validation at parse time of what has been assigned, however type checking is really not necessary since validation will occur in contexts when a variable is instantiated anyway. Another variable type (VarType) to introduce to the BNF grammar is "Generic" wherein anything assigned to the variable is to have its type delayed until after instantiation (i.e. when referenced later). Note that the str variable is not instantiated at this time (i.e. = the preferred embodiment, however an alternate embodiment would instantiate str at this time). Below will demonstrate a Generic variable instantiation.

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

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

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

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

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

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

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

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

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

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

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

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

"Accounting Department" [H;]=\track;

The grant "Accounting Department" is subordinate to "Work", has optional History information to be generated, and has one (1) privilege assigned.

Parents={Mom=\lbxall; Dad=\lbxall;};

Michael-Friends=\geoarr;\geode;

Jason-Friends=\nearar;\nearde;

The grant "Parents" is independent of the Work grant (a peer), has two (2) subordinate grants "Mom" and "Dad", each with a single privilege assigned. The grants "Michael-Friends" and "Jason-Friends" are each independent of other grants, and each have two (2) privileges assigned. A nested tree structure of Grants so far compiled which can be used for privilege assignments are:

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

The nested structure of the source code was intended to highlight the relationship of grants defined. Note that assigning the Work grant from one ID to another ID results in assigning all privileges of all subordinate grants (i.e. \geoar;\geodeMearar;\nearde;\mssys;\msbios;\msguiload;\msapps;\track).

Bill: LBXPHONE_USERS [G=\caller;\callee;\trkall;];

The MS ID Bill assigns (i.e. Grant specification "G") three (3) privileges to the LBXPHONE_USERS group (i.e. to each member of the group). Privileges and/or grants can be granted. The \caller privilege enables LBXPHONE_USERS member MSs to be able to call the Bill MS. The \callee privilege enables the Bill MS to call LBXPHONE_USERS member MSs. The \trkall privilege enables LBXPHONE_USERS members to use the MS local tracking application for reporting mobile whereabouts of the Bill MS. The grants are optional (i.e. "[" and "]") because without specific grants and/or privileges specified, all privileges are granted.

LBXPHONE_USERS: Bill [G=\callee;\caller;];

Each member of the LBXPHONE_USERS group assigns (i.e. Grant specification "G") two (2) privileges to the Bill MS. The \caller privilege enables the Bill MS to be able to call any of the members of the LBXPHONE_USERS group. The \callee privilege enables the LBXPHONE_USERS member MSs to call the Bill MS.

Bill:Sophia;

All system privileges are assigned from Bill to Sophia.

Bill:Brian [*assignPrivs];

The assignPrivs variable is instantiated to "G=Family,Work,\vuloc [T=>20080402000130.24,<20080428; D=*str; H;]" as though that configuration were made literally as:

Bill:Brian [G=Family,Work,\vuloc [T=>20080402000130.24,<20080428; D="Test Case #106729 (context)"; H;]];

Note the str variable is now instantiated as well. Bill grants Brian all privileges defined in the Family grant, all privileges of the Work grant, and the specific \vuloc privilege. The privilege \vuloc has optional specifications for TimeSpec (i.e. after 1 minute 30.24 seconds into Apr. 2, 2008 and prior to Apr. 28, 2008), Description, and History to be generated. The optional specifications ([. . .]) would have to be outside of the other optional delimiter specifications (e.g. [G= . . .] [.]) to be specifications for the Permission.

Bill:George [G=\geoall,\nearall;];

Bill assigns two (2) privileges to George.

Michael: Bill [G=Parents,Michael-Friends;];

Michael assigns to Bill the privileges \lbxall, \geoarr and \geode.

Jason: Bill [G=Parents,Jason-Friends;];

Jason assigns to Bill the privileges \lbxall, \nearar and \nearde.

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

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

FIG. **51**B shows that a Charters block contains "stuff" between delimiters ({, }) like PPLs, except the reserved keyword "Charters" qualifies the block which follows. Statements within the block are also aligned with syntax of PPLs. Here is an in-context description of FIG. **51**B:

Condition(cond1)="(_location @@ \loc_my) [D="Test Case #104223 (v)";]";

The variable cond1 is of type Condition and is set accordingly. Validation of the variable type can occur here since the type is known. Condi is a Condition specification with an optional specification for the Description. Since the type "Generic" can be used, it may convenient to always use that.

"ms group"={"Jane", "George", "Sally"};

This is another method for specifying a group without a Groups block. The internalizer preferably treats an assignment using block delimiters outside of any special block definitions as a group declaration. While there has been no group hierarchies demonstrated, groups within groups can certainly be accomplished like Grants.

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

_msid is a WDRTerm indicating to check the condition of the WDRs maintained to the local MS (e.g. processed for inserting to queue **22**). The condition _msid="Michael" tests if the WDR in process has a WDR MS ID field **1100***a* equal to the MS ID Michael. "&" is a CondOp. After instantiation of cond1 with the string substitution the second condition is "(_location @@ \loc_my) [D="Test Case #104223 (v)";]" which tests the WDR in process (e.g. for insertion to queue **22**) for a WDR location field **1100***c* which was at my current location (\loc_my is a system defined atomic term for "my current location" (i.e. the current location of the MS checking the WDR in process)). @@ is an atomic operator for "was at". There is an optional description specified for the condition to be generated. The expression formed on the left hand side of the colon (:) not only tests for Michael WDR information, but also Jason WDR information with the same WDR field tests. If the WDR in process (contains a MS ID=Michael AND Michael's location was at my current location at some time in the past), OR (i.e. |CondOp) the WDR in process (contains a MS ID=Jason AND Jason's location was at my current location at some time in the past), then the Actions construct (i.e. right hand side of colon) is acted upon. The "was at" atomic

operator preferably causes access to LBX History **30** after a fruitless access to queue **22**. It may have been better to specify another condition for Michael and Jason WDRs to narrow the search, otherwise if LBX history is not well pruned the search may be timely. For example, the variable may have been better defined prior to use as:

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

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

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

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

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

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

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

(M_sender = ~emailAddrVar [T=<YYYYMMDD18]):
Notify Indicator (M_sender, \thisms) [D="Test Case #104223"; H;];

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

a WDR that triggers checking the configuration seems useless, however a MS may have many WDRs in process thereby reasonably causing frequent checks to M_sender. A more useful charter with an AppTerm will check the AppTerm against a WDR field or subfield, while keeping in mind that WDRs in process trigger testing the charter(s). For example:

(_appfld.email.source=M_sender)

or the equivalent of:

(M_sender=_appfld.email.source)

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

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

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

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

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

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

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

The action demonstrates an interesting format for representing the optional Host construct (qualifier) of the BNF grammar for where the action should take place (assuming privilege to execute there is configured). "ms group"[G]. tells the internalizer to search for a group definition like an array and find the first member of the group meeting the subscript definition. This would be "George" (the G). Any substring of "George" (or the entire string) could have been used to indicate use George from the "ms group". This allows a shorthand reference to the item(s) of the group. Multiple members that match "G" would all apply for the action. Also, note that the double quotes are used whenever variables contain significant blanks. "ms group"[G].Store DBobject tells the internalizer that the Command Operand pair is to be executed at the George MS for storing to a database object per parameters. An equivalent form is George.Store DB-object with the Host specification explicitly specified as George. The parameters of (JOESDB.LBXTABS.TEST, "INSERT INTO TABLESAV ("'&& \thisMS &&"', "'&& \timestamp &&"', 9);", \thisMS) indicates to insert a row into the table TABLESAV of the TEST database at the system "this MS" (the MS hosting the configuration). The second (query) parameter matches the number of columns in the table for performing a database row insert. Like other compilers/interpreters, the " " evaluates to a single double quote character when double quotes are needed inside strings. A single quote can also be legal to delimit query string parameters (as shown). This example shows using atomic term(s) for a parameter (i.e. elaborates to underlying value; WDRTerm(s) can also be used for parameters). This example introduces a concatenation operator (&&) for concatenating together multiple values into a result string for one parameter (e.g. "INSERT INTO TABLESAV ('Bill', '20080421024421.45', 9);"). Other embodiments will support other programmatic operators in expressions for parameters. Still other embodiments will support any reasonable programmatic statements, operators, and syntax among charter configuration to facilitate a rich method for defining charters **12**.

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

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

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

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

_I_msid is a WDRTerm indicating to check the condition of the WDRs inbound to the local MS (e.g. deposited to receive queue **26**). The condition _I_msid="Sophia" tests if the

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

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

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

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

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

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

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

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

FIG. **53** depicts a preferred embodiment of a Prefix Registry Record (PRR) for discussing operations of the present disclosure. A PRR **5300** is for configuring which prefix is assigned to which application used in an AppTerm. This helps to ensure that an AppTerm be properly usable when referenced in a charter. A prefix field **5300***a* provides the prefix in an AppTerm syntax (e.g. M_sender such that "M" is the prefix). Any string can be used for a prefix (i.e. configured in field **5300***a*), but preferably there are a minimal number of characters to save syntax encoding space. A description field **5300***b* provides an optional user specified description for a PRR **5300**, but it may include defaulted data available with an application supporting at least one AppTerm. A service references field **5300***c* identifies, if any, the data processing system services associated with the application for the AppTerm referenced with the prefix of field **5300***a*. Validation of such services may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***c* potentially contains a list of service references. An application references field **5300***d* identifies, if any, data processing system application references (e.g. names) associated with the Application for the AppTerm referenced with the prefix of field **5300***a*. Validation of such applications referenced may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***d* potentially contains a list. A process references field **5300***e* identifies, if any, data processing operating system

processes for spawning associated with the Application for the AppTerm referenced with the prefix of field **5300***a*. Validation of such processes may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***e* potentially contains a list. A paths field **5300***f* identifies, if any, data processing system file name paths to executables (e.g. .exe, .dll, etc) for spawning associated with the Application for the AppTerm referenced with the prefix of field **5300***a*. Validation of such paths may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***f* potentially contains a list. A documentary field **5300***g* documents each Application data variable (i.e. AppTerm data name without prefix), and an optional description, for what data is exposed for the Application which can be used in the AppTerm. Validation of data in field **5300***g* data may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***g* potentially contains a list. Extension field **5300***h* contains other data for how to test for whether or not the Application of the PRR is up and running in the MS, additional information for starting the Application, and additional information for accessing application vitals. Validation of information may occur through an API, or may be specified by a knowledgeable user, administrator, or system setup. Field **5300***h* may be a list, or null.

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

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

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

FIG. **55**A depicts a flowchart for describing a preferred embodiment of MS user interface processing for Prefix Registry Record (PRR) configuration. Block **5502** may begin as the result of an authenticated administrator user interface, authenticated user interface, or as initiated by a user. Block **5502** starts processing and continues to block **5504** where initialization is performed before continuing to block **5506**. Initialization may include initializing for using an SQL database, or any other data form of PRRs. Processing continues to block **5506** where a list of current PRRs are presented to the user. The list is scrollable if necessary. A user preferably has the ability to perform a number of actions on a selected/specified PRR from the list presented at block **5506**. Thereafter, block **5508** waits for a user action in response to presenting PRRs. Block **5508** continues to block **5510** when a user action has been detected. If block **5510** determines the user selected to modify a PRR, then the user configures the specified PRR at block **5512** and processing continues back to block **5506**. Block **5512** interfaces with the user for PRR **5300** alterations until the user is satisfied with changes which may or may not have been made. Block **5512** preferably validates to the fullest extent possible the data of PRR **5300**. If block **5510** determines the user did not select to modify a PRR, then processing continues to block **5514**. If block **5514** determines the user selected a PRR for delete, then block **5516** deletes the specified PRR, and processing continues back to block **5506**. Depending on an embodiment, block **5516** may also properly terminate the application fully described by the PRR **5300**. If block **5514** determines the user did not select to delete a PRR, then processing continues to block **5518**. If block **5518** determines the user selected to add a PRR, then the user adds a validated PRR at block **5520** and processing continues back to block **5506**. Block **5520** preferably validates to the fullest extent possible the data of PRR **5300**. Depending on an embodiment, block **5520** may also properly start the application described by the PRR **5300**. If block **5518** determines the user did not select to add a PRR, then processing continues to block **5522**. If block **5522** determines the user selected to show additional detail of a PRR, then block **5524** displays specified PRR details including those details not already displayed at block **5506** in the list. Processing continues back to block **5506** when the user is complete browsing details. If block **5522** determines the user did not want to browse PRR details, then processing continues to block **5526**. If block **5526** determines the user selected to enable/disable (toggle) a specified PRR, then block **5528** uses PRR **5300** to determine if the associated application is currently enabled (e.g. running) or disabled (e.g. not running). Upon determination of the current state of the application for the specified PRR **5300**, block **5528** uses the PRR **5300** to enable (e.g. start if currently not running)), or disable (e.g. terminate if currently running), the application described fully by the specified PRR, before continuing back to block **5506**. Block **5528** should ensure the Application has been properly started, or terminated, before continuing back to block **5506**. If block **5526** determines the user did not want to toggle (enable/disable) a PRR described application, then processing continues to block **5530**. If block **5530** determines the user selected to display candidate AppTerm supported applications of the MS, then block **5532** presents a list of MS applications potentially configurable in PRR form. Block **5532** will interface with the user until complete browsing the list. One embodiment of block **5532** accesses current PRRs **5300** and displays the applications described. Another embodiment accesses an authoritative source of candidate AppTerm supported applications, any of which can be configured as a PRR. Processing continues back to block **5506** when the user's browse is complete. If block **5530** determines the user did not select to display AppTerm supported applications, then processing continues to block **5534**. If block **5534** determines the user selected to use a data source as a template for automatically populating PRRs **5300**, then block **5536** validates a user specified template, uses the template to alter PRRs **5300**, and processing continues back to block **5506**. PRRs may be optionally altered at block **5536** for replacement, overwrite, adding to, or any other alternation method in accordance with a user or system preference. In some embodiments, existing PRRs can be used for template(s). If block **5534** determines the user did not select to use a data source for a PRR template, then processing continues to block **5538**. If block **5538** determines the user

did not select to exit PRR configuration processing, then block **5540** handles all other user actions detected at block **5508**, and processing continues back to block **5506**. If block **5538** determines the user did select to exit, then processing continues to block **5542** where configuration processing cleanup is performed before terminating FIG. **55**A processing at block **5544**. Depending on an embodiment, block **5542** may properly terminate data access initialized at block **5504**, and internalize PRRs for a well performing read-only form accessed by FIG. **55**B. Appropriate semaphore interfaces are used.

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

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

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

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

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

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

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

Blocks **5640** and **5650** preferably have a stringVar set to the permission/charter data encoding start position, and then set a length of the permission/charter data for processing by block **5642**. Alternatively, the stringVar is a null terminated string for processing the permission(s)/charter(s) data encoding. Embodiment requirements are for providing appropriate parameters for invoking block **5642** for unambiguous processing of the entire permission(s)/charter(s) for parsing and processing. The procedure of block **5642** has already been described throughout this disclosure (e.g. creating a processable internalized form (e.g. database records, programmatic structure, etc)). Upon return from block **5642** processing, block **5644** resets the parsing position of the data source encoding provided at block **5602** for having already processed the permission(s)/charter(s) encoding handled by block **5642**. Thereafter, processing continues back to block **5632** for getting the next token from the data encoding source.

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

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

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

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

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

LBX: Permissions and Charters—WDR Processing

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

mWITS:

- The preferred embodiment is a new block **273** in FIG. **2**F such that block **272** continues to block **273** and block **273** continues to block **274**. This allows mWITS processing block **273** to see all WDRs which are candidate for insertion to queue **22**, regardless of the role check at block **274**, confidence check at block **276**, and any other FIG. **2**F processing. In some embodiments, block **273** may choose to use enabled roles and/or confidence and/or any WDR field(s) values and/or permissions and/or any other processing result to decisively affect whether or not the WDR should be examined and/or processed further by FIG. **2**. For example, block **273** may result in processing to continue directly to block **294** or **298** (rather than block **274**). For example, upon determining that the WDR source had not provided any privileges to the receiving MS, the WDR can be ignored so as to not use resources of the MS. In another example, a WDR shows that it arrived completely wirelessly (e.g. field(s) **1100***f*) and did not go through an intermediary service (e.g. router). The WDR may provide usefulness in locating the receiving MS despite the receiving MS not being privileged by the source MS, in which case block **273** continues to block **274** for WDR processing. It may be important to filter WDRs so that only those WDRs are maintained which either a) contribute to locating (per configurations), or b) are associated with active permissions or charters for applicable processing. The WRC discussed above may also be used to cause block **273** to continue to block **294** or **298**. Such filtering is referred to as WITS filtering. WITS filtering may be crucial in a LBX architecture which supports MSs great distances from each other since there can be an overloading number of WDRs to process at any point in time. Charters and privileges that are configured are used for deciding which WDRS are to be "seen" (processed) further by FIG. **2**F processing. If there are no privileges and no charters in effect for the in process WDR, then the WDR may be ignored. If there is no use for the WDR to help locate the receiving MS, then the WDR may also be ignored. If there are privileges and charters in effect for the in process WDR, then the WDR can be processed further by FIG. **2**F, even if not useful for locating the MS.
- One preferred embodiment does make use of the confidence field **1100***d* to ensure the peer MS has been sufficiently located. Block **273** will compare information of the WDR with configured privileges to determine which actions should be performed. When appropriate privileges are in place, block **273** will also compare information of the WDR with configured and privileged charters (e.g. _fldname) to determine applicable configured charter actions to be performed.
- Alternate embodiments can move mWITS at multiple processing places subsequent to where a WDR is completed by the MS (e.g. blocks **236**, **258**, **334**, **366**, **418**, **534**, **618**, **648**, **750**, **828**, **874**, **958**, **2128**, **2688**, etc).
- Another embodiment can support mWITS at processing places subsequent to processing by blocks **1718** and **1722** to reflect user maintenance.
- Yet another embodiment recognizes in mWITS that the WDR was first inbound to the MS and is now in process of being maintained (e.g. to queue **22**). This can allow distinguishing between an inbound WDR, maintained WDR, and inbound AND maintained WDR. In one embodiment, the WDR (e.g. field **1100***g*) carries new bit(s) of information (e.g. set by receive processing when inserting to queue **26**) for indicating the WDR was inbound to the MS. The new bit(s) are checked by mWITS for new processing (i.e. inbound AND maintained WDR).

iWITS:

- The preferred embodiment is a new block **2111** in FIG. **21** such that block **2110** continues to block **2111** (i.e. on No

condition) and block **2111** continues to block **2112**. This allows iWITS processing block **2111** to see all inbound WDRs, regardless of the confidence check at block **2114**, and any other FIG. **21** processing. In some embodiments, block **2111** may choose to use confidence and/or any WDR field(s) and/or permissions and/or any other processing result to decisively affect whether or not the WDR should be examined and/or processed further by FIG. **21**. Block **2111** may result in processing to continue directly to block **2106** (rather than block **2112**). For example, upon determining that the WDR source had not provided any privileges to the receiving MS, the WDR can be ignored so as to not use resources of the MS. In another example, a WDR shows that it arrived completely wirelessly (e.g. field(s) **1100***f*) and did not go through an intermediary service (e.g. router). The WDR may provide usefulness in locating the receiving MS despite the receiving MS not being privileged by the source MS, in which case block **2111** continues to block **2112** for WDR processing. Similar WITS filtering can occur here as was described for mWITS processing above, with the advantage of intercepting WDRs of little value at the earliest possible time and preventing them from reaching subsequent LBX processing.

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

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

oWITS:

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

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

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

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

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

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

Block **5700** continues to block **5702**-*a* where the WRC and applicable origination information of the WDR is accessed. Thereafter, if the WRC and WDR information indicates to ignore the WDR at block **5702**-*b*, then processing continues to block **5746**, otherwise processing continues to block **5704**. Whenever block **5746** is encountered, the decision is made (assumed in FIG. **57**) to continue processing the WDR or not continue processing the WDR in processing which includes

FIG. **57** (i.e. FIGS. **2**F, **20**, **21** **25**) as described above. This decision depends on how block **5746** was arrived to by FIG. **57** processing.

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

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

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

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

Different identity embodiments are supported (e.g. MS ID or user ID) for the LHS and/or RHS (see BNF grammar for different embodiments). A privilege preferably grants the ability to create effective (enabled) charters for one ID from another ID. However, in some embodiments the granting of a charter by itself from one ID to another ID can be treated like the granting of a permission/privilege to use the charter, thereby preventing special charter activating permission(s) be put in place. Charter data collection **5852** is also to be from the perspective of the MS of FIG. **57** processing. Thus, the terminology "this MS ID" refers to the MS ID of the MS of FIG. **57** processing. The terminology "WDR MS ID" is the MS ID (field **1100***a*) of the in-process WDR of FIG. **57** processing distinguished from all other MS IDs configured in collection **5852** at the time of processing the WDR. The terminology "other MS IDs" is used to distinguish all other MS IDs configured in collection **5852** which are not the same as the MS ID of the terminology "WDR MS ID" (i.e. MS IDs other than the MS ID (field **1100***a*) of the in-process WDR of FIG. **57** processing (also other than the "this MS" MS ID)). Charter configurations **5860** are charters created by the MS ID of an in-process WDR (i.e. WDR being processed by FIG. **57** at "this MS") for being effective at the MS of FIG. **57** processing (this MS ID). The groups an ID belongs to can also provide, or be provided with, charters so that the universe of charters granted should consider groups as well. Charter configurations **5870** are charters created by the MS ID of FIG. **57** processing (i.e. this MS) for being effective at the MS of FIG. **57** processing (this MS ID). Charter configurations **5870** include the most common embodiments of creating charters for yourself at your own MS. Charter configurations **5880** are charters created by the MS ID of FIG. **57** processing (this MS) for being effective at MSs with MS IDs configured in collection **5852** other than the MS ID of the in-process WDR being processed by FIG. **57**. Charter configurations **5890** are charters at the MS of FIG. **57** processing (this MS) which are created by MS IDs other than the MS ID of the in-process WDR being processed by FIG. **57** (also other than the "this MS" MS ID).

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

in-process embodiments when considering generally "incoming" WDRs (i.e. being maintained (e.g. to queue **22**) or being inbound (e.g. communicated to the MS)).

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

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

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

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

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

With reference now to FIG. **59**, depicted is a flowchart for describing a preferred embodiment of a procedure for enabling LBX features and functionality in accordance with a certain type (category) of permissions. Blocks **5920**, **5924**, **5928**, **5932**, **5936**, **5940**, **5944**, and **5946** enable or disable LBX features and functionality for semantic privileges. Processing of block **5734** starts at block **5900** and continues to block **5902** where the permission type list parameter passed (i.e. PRIVS2ME (**5810**) when invoked from block **5734**) is determined, and the in-process WDR may be accessed. The list parameter passed provides not only the appropriate list to FIG. **59** processing, but also which list configuration (**5810**, **5820**, **5830** or **5840**) has been passed for processing by FIG. **59**. There are potentially thousands of specific privileges that FIG. **59** can handle. Therefore, FIG. **59** processing is shown to generically handle different classes (categories) of privileges, namely privilege classes of: privilege-configuration, charter-configuration, data send, impersonation, WDR processing, situational location, monitoring, LBX, LBS, and any others as handled by block **5946**. Privileges disclosed throughout the present disclosure fall into one of these classes handled by FIG. **59**.

Block **5902** continues to block **5904** where if it is determined that a privilege-configuration privilege is present in the list parameter passed to FIG. **59** processing, then block **5906** will remove privileges from the list parameter if appropriate to do that. For example, a privilege (or absence thereof) detected in the list parameter for indicating no privileges can be defined/enabled in context of the list parameter causes block **5906** to remove all privileges from the list parameter and also from permissions **10** (i.e. **5810** of collection **5802** when FIG. **59** invoked from block **5734**). Similarly, any more granular privilege-configuration privileges of the list parameter causes processing to continue to block **5906** for ensuring remaining privileges of the list parameter (and of permissions

10 configurations) are appropriate. There can be many different privilege-configuration privileges for what can, and can't, be defined in permissions **10**, for example by any characteristic(s) of permissions data **10** according to the present disclosure BNF grammar. Block **5906** continues to block **5908** when all privilege-configuration privileges are reflected in the list parameter and collection **5802** of permissions **10**. If block **5904** determines there are no privilege-configuration privileges to consider in the list parameter passed to FIG. **59** processing, then processing continues to block **5908**.

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

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

If block **5922** determines the entry is an impersonation privilege, then block **5924** will enable LBX features and functionality appropriately in context for the list parameter, and processing continues back to block **5908**. An impersonation privilege is one that is used to access certain authenticated user interfaces, some of which were described above. Any granulation of permission data **10** (e.g. by any characteristic(s)) may be supported, for example for any subset of MS user interfaces with respect to the present disclosure. Block **5924** may access security, or certain application interfaces accessible to the MS of FIG. **59** processing for read, modify, add, or otherwise alter certain related data, or cause the processing of certain related executable code, for example to manage associated identity impersonation at the MS. If block **5922** determines the entry is not an impersonation privilege, then processing continues to block **5926**. Impersonation privileges can be overall privileges, subordinate privileges, and/or privileges for any granulation of identity data or any other characteristic(s) available from a derivative of the BNF grammar of FIGS. **30**A through **30**E.

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

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

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

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

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

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

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

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

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

With reference now to FIG. **60**, depicted is a flowchart for describing a preferred embodiment of a procedure for performing LBX actions in accordance with a certain type of permissions. Blocks **6012**, **6016**, **6020**, **6024**, **6028**, **6032**, **6036**, and **6038** perform actions for semantic privileges. Processing of block **5736** starts at block **6002** and continues to block **6004** where the permission type parameter passed (i.e. PRIVS2ME (**5810**) when invoked from block **5736**) is deter-

mined, and the in-process WDR may be accessed. The list parameter passed provides not only the appropriate list to FIG. **60** processing, but also which list configuration (**5810**, **5820**, **5830** or **5840**) has been passed for proper processing by FIG. **60**. There are potentially thousands of specific privileges that FIG. **60** can handle. Therefore, FIG. **60** processing is shown to generically handle different classes (categories) of privileges, namely privilege classes of: data send, impersonation, WDR processing, situational location, monitoring, LBX, LBS, and any others as handled by block **6038**. Privileges disclosed throughout the present disclosure fall into one of these classes handled by FIG. **60**.

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

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

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

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

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

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

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

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

Block **6038** is provided for processing completeness for handling appropriately (e.g. performing any LBX actions in context for the list parameter (if any applicable) a privilege that some reader may not appreciate falling into one of the privilege classes of FIG. **60** processing. Block **6038** then continues to block **6006**. Referring back to block **6008**, if it is determined there are no more unprocessed entries remaining in the list parameter (i.e. **5810** of collection **5802** when FIG. **60** invoked from block **5736**), then the caller/invoker is returned to at block **6040**.

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

FIG. **60** processing causes application types of actions according to privileges set. Such application types of actions are preferably caused using APIs, callback functions, or other interfaces so as to isolate FIG. **60** LBX processing from applications that are integrated with it. This prevents application "know-how" from being part of the LBX processing (e.g. software) built for MSs. FIG. **60** preferably invokes the "know-how" through an appropriate interface (software or hardware). In one preferred embodiment, participating applications register themselves as processing particular atomic privileges so that FIG. **60** invokes the interface with the privilege, its setting, and perhaps useful environmental data of interest. The application itself can then optimally process the privilege for an appropriate application action. Invocation of the application interface may be thread oriented so as to not wait for a return, or may be synchronous for waiting for a

return (or return code). In one preferred embodiment, the PRR **5300** is modified for further containing a privilege join field **5300***j* for joining to a new Application Privileges Reference (APR) table containing all privileges which are relevant for the application described by the PRR **5300**. This provides the guide of all privileges which are applicable to an application, and which are to cause invocation of the interface(s) of the application. A PRR **5300** is to be extended with new data in at least one field **5300***k* which contains interface directions for how to invoke the application with the privilege for processing (e.g. through a Dynamic Link Library (DLL), or script, interface). Preferably, a single API or invocation is used for all privileges to a particular application and the burden of conditional processing paths is put on the application in that one interface. An alternate embodiment could allow multiple interfaces to be plugged in: one for each of a plurality of classes, or categories, of privileges so that the burden of unique processing paths, depending on a privilege, is reduced for one application. In any embodiment, it is preferable to minimize linkage execution time between LBX processing and an application which is plugged in. Linkage time can be reduced by:

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

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

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

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

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

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

With reference back to block **5706**, if it is determined that the WDR identity matches the MS of FIG. **57** processing, processing continues to block **5748**. Block **5706** continues to block **5748** when a) the in-process WDR is from this MS and is being maintained at the MS of FIG. **57** processing (i.e. FIG. **57**=mWITS); or b) the in-process WDR is outbound from this MS (i.e. FIG. **57**=oWITS). Block **5748** forms a PRIVS2OTHERS list of configurations **5830** and continues to block **5750** for eliminating duplicates that may be found. Block **5748** may collapse grant hierarchies to form the list. Duplicates may occur for privileges which include the duplicated privileges (i.e. subordinate privileges) as described above. Block **5750** additionally eliminates duplicates that may exist for permission embodiments wherein a privilege can enable or disable a feature. In a present disclosure embodiment wherein a privilege can enable, and a privilege can disable the same feature or functionality, there is preferably a tie breaker of disabling the feature (i.e. disabling wins). In an alternate embodiment, enabling may break a tie of ambiguity. Block **5750** further eliminates privileges that have a MSRelevance qualifier indicating the MS of FIG. **57** processing is not supported for the particular privilege, and also eliminates privileges with a TimeSpec qualifier invalid for the time of FIG. **57** processing (an alternate embodiment can enforce TimeSpec interpretation at block **5758** (i.e. in FIG. **60** processing)). Thereafter, block **5752** forms a MYCHARTERS list of configurations **5870** and preferably eliminates variables by instantiating/elaborating at points where they are referenced. Then, block **5754** forms a CHARTERS2ME list of configurations **5890** and preferably eliminates variables by instantiating/elaborating at points where they are referenced. Then, block **5756** eliminates those charters which are not privileged. In some embodiments, block **5756** is not necessary (**5754** continues to **5758**) because un-privileged charters will not be permitted to be present at the MS of FIG. **57** processing. Nevertheless, block **5756** removes from the CHARTERS2ME list all charters which do not have a privilege granted by the MS (the MS user) of FIG. **57** processing to the creator of the charter, for permitting the charter to be

enabled (as described above for block **5718**). In any embodiments, block **5756** ensures no charters from other users are considered active unless appropriately privileged. Thereafter, block **5758** performs actions in accordance with privileges of the PRIVS2OTHERS list by invoking the do action procedure of FIG. **60** with the PRIVS2ME list, and the in-process WDR as parameters (preferably passed by pointer/reference), and then continues to block **5740** which has already been described. FIG. **60** processing is the same as described above except in context for the PRIVS2OTHERS (**5830**) and for the in-process WDR of FIG. **57** processing relative the PRIVSOTHERS list. Of course the context of blocks **5748** through **5758** are processed for in-process WDRs which are: a) maintained to the MS of FIG. **57** for the whereabouts of the MS of FIG. **57** processing; or b) outbound from the MS of FIG. **57** processing (e.g. an outbound WDR describing whereabouts of the MS of FIG. **57** processing). One embodiment may incorporate a block **5757** (block **5756** continues to **5757** which continues to block **5758**) for invoking FIG. **59** processing with PRIVS2OTHERS. Generally, privilege configurations **5830** involve actions for the benefit of others (i.e. other than this MS).

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

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

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

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

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

FIG. **57** determines which privileges and charters are relevant to the WDR in process, regardless of where the WDR originated. The WDR identity checked at block **5706** can take on various embodiments so that the BNF grammar of FIGS. **30**A through **30**E are fully exploited. Preferably, the identities associated with "this MS" and the WDR in process are usable as is, however while there are specific embodiments implementing the different identifier varieties, there may also be a translation or lookup performed at block **5704** to ensure a proper compare at block **5706**. The identities of "this MS" and the WDR identity (e.g. field **1100***a*) may be translated prior to performing a compare. For example, a user identifier maintained to the user configurations (permissions/charters) may be "looked up" using the MS identifiers involved ("this MS" and WDR MS ID) in order to perform a proper compare at block **5706**. Some embodiments may maintain a separate

identifier mapping table local to the MS, accessed from a remote MS when needed, accessed from a connected service, or accessed as is appropriate to resolve the source identifiers with the identifiers for comparing at block **5706**. Thus, permissions and/or charters can grant from one identity to another wherein identities of the configuration are associated directly (i.e. useable as is) or indirectly (i.e. mapped) to the actual identities of the user(s), the MS(s), the group(s), etc involved in the configuration.

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

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

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

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

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

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

Referring back to block **6112**, if it is determined there are no special terms in the expression remaining to process (or there were none in the expression), then block **6122** evaluates the expression to a Boolean True or False result using well known processing for a stack based parser for expression evaluation (e.g. See well known compiler/interpreter development techniques (e.g. "Algorithms+Data Structures=Programs" by Nicklaus Wirth published by Prentice-Hall, Inc. 1976)). Block **6122** implements atomic operators using the WDR queue **22**, most recent WDR for this MS, LBX history **30**, or other suitable MS data. Any Invocation is also invoked for resulting to a True or False wherein a default is enforced upon no return code, or no suitable return code, returned. Invocation parameters that had special terms would have been already been updated by block **6118** to eliminate special terms prior to invocation. Thereafter, if block **6124** determines the expression evaluated to False, then processing continues back to block **6104** for the next charter (i.e. expression=False implies to prevent (not cause) the action(s) of the charter). If block **6124** determines the expression evaluated to True, then processing continues to block **6126**.

Block **6126** begins an iterative loop (blocks **6126** through **6162**) for processing all actions from the current charter. Block **6126** gets the next (or first) action (if any) from the charter and continues to block **6128**. There should be at least one action in a charter provided to FIG. **61** processing since the preferred embodiment of FIG. **57** processing will have eliminated any placeholder charters without an action specified (e.g. charters with no actions preferably eliminated at blocks **5740** as part of the merge process, at block **5742**, or as part of previous FIG. **57** processing to form privileged charter lists). If block **6128** determines an unprocessed action was found for processing, then block **6130** initializes a REMOTE

variable to No. Thereafter, if it is determined at block **6132** that the action has a BNF grammar Host specification, then block **6134** accesses privileges and block **6136** checks if the action is privileged for being executed at the Host specified. The appropriate permissions **5802** are accessed at block **6134** in this applicable context of FIG. **57** processing. If block **6136** determines the action is privileged for running at the Host, then block **6138** sets the REMOTE variable to the Host specified and processing continues to block **6140**. If block **6136** determines the action is not privileged for running at the Host, then processing continues to block **6120** for error processing already described above. If block **6132** determines there was no Host specified for the action, processing continues directly to block **6140**. Blocks **6134** and **6136** may not be required since unprivileged charters were already eliminated in previous blocks of FIG. **57** processing (e.g. see blocks **5718** and **5756**). Nevertheless, blocks **6134** and **6136** are shown to cover other embodiments, and to ensure unprivileged charters are treated ineffective. Depending on an embodiment, blocks **5718** and **5756** may only perform obvious eliminations. In other embodiments, there may be no blocks **5718** or **5756** so that charter part processing occurs only in one place (i.e. FIG. **61**) to achieve better MS performance by preventing more than one scan over charter data. In another embodiment, blocks **6134** and **6136** are not required since all charter eliminations based on privileges already occurred at the previous blocks of FIG. **57** processing. Block **6132** can continue to block **6138** when blocks **6134** and **6136** are not required and a Host was specified with the action.

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

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

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

Referring back to block **6150**, if it is determined that the special term of the parameter list is not privileged, then processing continues to block **6120** for error processing already described. Referring back to block **6146**, if it is determined there are no special terms in the parameter list remaining to process (or there were none), then block **6154** evaluates each and every parameter expression to a corresponding value using well known processing for a stack based parser for expression evaluation (e.g. See well known compiler/interpreter development techniques (e.g. "Algorithms+Data Structures=Programs" by Nicklaus Wirth published by Prentice-Hall, Inc. 1976)). Block **6154** implements the atomic operators using the WDR queue **22**, most recent WDR for this MS, LBX history **30**, or other suitable MS data. Any Invocation is also invoked for resulting to Data or Value wherein a default is enforced upon no returned data. Invocation parameters that had special terms would have been updated at block **6152** to eliminate special terms prior to invocation. Block **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. **75**A for performing the action at the remote MS (also see FIG. **75**B). 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. **75**A, 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. **75**A is preferably of linkable PIP code **6**. The purpose is for the MS of FIG. **75**A 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. **75**A (e.g. block **6162**), there can be intelligence to drive each channel for broadcasting on multiple channels, either by multiple send threads for FIG. **75**A processing, FIG. **75**A loop processing on a channel list, and/or passing channel information to send processing feeding from queue **24**. If FIG. **75**A does not transmit directly over the channel(s) (i.e. relies on send processing feeding from queue **24**), an embodiment may provide means for communicating the channel for broadcast/send processing when interfacing to queue **24** (e.g. incorporate a channel qualifier field with send packet inserted to queue **24**).

In any case, see detailed explanations of FIGS. **13**A through **13**C, as well as long range exemplifications shown in FIGS. **50**A through **50**C, respectively. Processing begins at block **7502**, continues to block **7504** where the caller parameter(s) passed to FIG. **75**A 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. **75**B). 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. **13**A through **13**C, as further explained by FIGS. **50**A through **50**C 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. **75**A 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 19xx process. The receive execution data (RxED) process is to fit identically into the framework of architecture **1900** as other 19xx 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, param-

eters) 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 command 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 priority, 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 multithreaded/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

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 methodolo-

gies, 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 MS4 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 process-

ing). 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><I>) 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, _appfld.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 mW ITS, 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\benchmark.xml>=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 19xx 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 (requestors) 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 requestor 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, comprising:

accepting user input, from a user of a mobile application user interface of a user carried mobile data processing system, for configuring a user specified location based event configuration to be monitored and triggered by the mobile data processing system wherein the mobile data processing system uses the user specified location based event configuration to perform mobile data processing system operations comprising:

accessing at least one memory storing a first identifier and a second identifier and a third identifier wherein each identifier is determined by the mobile data processing system for at least one location based condition monitored by the mobile data processing system for the mobile data processing system triggering a location based action, the location based action performed by the mobile data processing system upon the mobile data processing system determining the at least one location based condition including whether identifier data determined by the mobile data processing system for a wireless data record received for processing by the mobile data processing system matches the third identifier and at least one of the first identifier and the second identifier, the wireless data record corresponding to a beaconed broadcast wireless data transmission that is beaconed outbound from an originating data processing system to a destination data processing system, the first identifier indicative of the mobile data processing system of the mobile application user interface for use by the mobile data processing system in comparing the first identifier to the identifier data determined by the mobile data processing system for the wireless data record received for processing by the mobile data processing system, the second identifier indicative of originating data processing system identity data of the wireless data record received for processing for use by the mobile data processing system in comparing the second identifier to the identifier data determined by the mobile data processing system for the wireless data record received for processing by the mobile data processing system, the third identifier indicative of the originating data processing system of the wireless data record received for processing wherein the third identifier is monitored by the mobile data processing system for use by the mobile data processing system in comparing the third identifier to the wireless data record received for processing by the mobile data processing system;

receiving for processing the wireless data record corresponding to the beaconed broadcast wireless data transmission that is beaconed outbound from the originating data processing system to the destination data processing system;

determining the identifier data for the wireless data record received for processing by the mobile data processing system;

comparing the identifier data for the wireless data record received for processing by the mobile data processing system with the third identifier and the at least one of the first identifier and the second identifier;

determining the at least one location based condition of the user specified location based event configuration including whether the identifier data for the wireless data record received for processing by the mobile data processing system matches the third identifier and the at least one of the first identifier and the second identifier; and

performing, upon the determining the at least one location based condition, the location based action in accordance with the determining the at least one location based condition of the user specified location based event configuration including whether the identifier data for the wireless data record received for processing by the mobile data processing system matches the third identifier and the at least one of the first identifier and the second identifier.

2. The method of claim **1** wherein the second identifier grants the location based action to the first identifier.

3. The method of claim 2 wherein the identifier data matches the second identifier and the location based action provides a location based feature intended for benefiting a data processing system user or group associated with the second identifier.

4. The method of claim 2 wherein the identifier data matches the first identifier and the location based action provides a location based feature intended for benefiting a data processing system user or group associated with the second identifier.

5. The method of claim 2 wherein the identifier data matches the second identifier and the location based action provides a location based feature intended for benefiting a data processing system user or group associated with the first identifier.

6. The method of claim 1 wherein the first identifier grants the location based action to the second identifier.

7. The method of claim 6 wherein the identifier data matches the second identifier and the location based action provides a location based feature intended for benefiting a data processing system user or group associated with the second identifier.

8. The method of claim 6 wherein the identifier data matches the first identifier and the location based action provides a location based feature intended for benefiting a data processing system user or group associated with the second identifier.

9. The method of claim 6 wherein the identifier data matches the second identifier and the location based action provides a location based feature intended for benefiting a data processing system user or group associated with the first identifier.

10. The method of claim 1 wherein the first identifier grants the location based action to the first identifier, and wherein the identifier data matches the first identifier and the location based action provides a location based feature intended for benefiting a data processing system user or group associated with the first identifier.

11. The method of claim 1 including:

accepting user input, from the user of the mobile data processing system, for administrating an other user specified location based event configuration including the first identifier and the second identifier, wherein the first identifier grants to the second identifier an other location based action to be performed by an other mobile data processing system, the second identifier being associated to the other mobile data processing system; and

communicating the other user specified location based event configuration to the other mobile data processing system for effectual use at the other mobile data processing system.

12. The method of claim 11 wherein the first identifier is administrated for matching identifier data associated with a wireless data record processed by the other mobile data processing system and the other location based action provides a location based feature intended for benefiting a data processing system user or group associated with the second identifier.

13. The method of claim 11 wherein the first identifier is administrated for matching identifier data associated with a wireless data record processed by the other mobile data processing system and the other location based action provides a location based feature intended for benefiting a data processing system user or group associated with the first identifier.

14. The method of claim 11 wherein the second identifier is administrated for matching identifier data associated with a wireless data record processed by the other mobile data processing system and the other location based action provides a location based feature intended for benefiting a data processing system user or group associated with the first identifier.

15. The method of claim 1 wherein the receiving for processing the wireless data record includes receiving for processing the wireless data record prior to transmitting the wireless data record outbound from the mobile data processing system.

16. The method of claim 1 wherein the receiving for processing the wireless data record includes receiving for processing the wireless data record after inbound receipt by the mobile data processing system of the wireless data record.

17. The method of claim 1 wherein the receiving for processing the wireless data record includes receiving for processing the wireless data record for insertion to a historical collection of information, the historical collection of information containing information for a plurality of data processing systems detected by the mobile data processing system to have been in a wireless vicinity of the mobile data processing system.

18. The method of claim 1 wherein the user specified location based event configuration includes a time or distance specification describing how the user specified location based event configuration is in effect.

19. The method of claim 1 wherein the user specified location based event configuration can be cloned by a user of an other mobile data processing system for subsequent use by the user of the other mobile data processing system.

20. The method of claim 1 wherein the first identifier or the second identifier or the third identifier or the identifier data identifies a group.

21. The method of claim 1 wherein a user associated to a fourth identifier has an impersonation privilege for administrating the user specified location based event configuration.

22. The method of claim 1 wherein the first identifier and the second identifier and the third identifier are each matched by the mobile data processing system to the identifier data.

23. The method of claim 1 wherein the user specified location based event configuration grants at least one semantic privilege or at least one grammar specification privilege.

24. A user carried mobile data processing system, comprising:

one or more processors; and

memory coupled to the one or more processors, wherein the memory includes executable instructions which, when executed by the one or more processors, results in the mobile data processing system accepting user input, from a user of a mobile application user interface of the mobile data processing system, for configuring a user specified location based event configuration to be monitored and triggered by the mobile data processing system wherein the mobile data processing system uses the user specified location based event configuration to perform mobile data processing system operations comprising:

accessing at least one memory storing a first identifier and a second identifier and a third identifier wherein each identifier is determined by the mobile data processing system for at least one location based condition monitored by the mobile data processing system for the mobile data processing system triggering a location based action, the location based action performed by the mobile data processing system upon the mobile data processing system determining the at least one location based condition including whether identifier data determined by the mobile data processing system for a wireless data record received for processing by the mobile data processing system

matches the third identifier and at least one of the first identifier and the second identifier, the wireless data record corresponding to a beaconed broadcast wireless data transmission that is beaconed outbound from an originating data processing system to a destination data processing system, the first identifier indicative of the mobile data processing system of the mobile application user interface for use by the mobile data processing system in comparing the first identifier to the identifier data determined by the mobile data processing system for the wireless data record received for processing by the mobile data processing system, the second identifier indicative of originating data processing system identity data of the wireless data record received for processing for use by the mobile data processing system in comparing the second identifier to the identifier data determined by the mobile data processing system for the wireless data record received for processing by the mobile data processing system, the third identifier indicative of the originating data processing system of the wireless data record received for processing wherein the third identifier is monitored by the mobile data processing system for use by the mobile data processing system in comparing the third identifier to the wireless data record received for processing by the mobile data processing system;

receiving for processing the wireless data record corresponding to the beaconed broadcast wireless data transmission that is beaconed outbound from the originating data processing system to the destination data processing system;

determining the identifier data for the wireless data record received for processing by the mobile data processing system;

comparing the identifier data for the wireless data record received for processing by the mobile data processing system with the third identifier and the at least one of the first identifier and the second identifier;

determining the at least one location based condition of the user specified location based event configuration including whether the identifier data for the wireless data record received for processing by the mobile data processing system matches the third identifier and the at least one of the first identifier and the second identifier; and

performing, upon the determining the at least one location based condition, the location based action in accordance with the determining the at least one location based condition of the user specified location based event configuration including whether the identifier data for the wireless data record received for processing by the mobile data processing system matches the third identifier and the at least one of the first identifier and the second identifier.

25. The user carried mobile data processing system of claim **24** wherein the second identifier grants the location based action to the first identifier.

26. The user carried mobile data processing system of claim **25** wherein the identifier data matches the second identifier and the location based action provides a location based feature intended for benefitting a data processing system user or group associated with the second identifier.

27. The user carried mobile data processing system of claim **25** wherein the identifier data matches the first identifier and the location based action provides a location based feature intended for benefitting a data processing system user or group associated with the second identifier.

28. The user carried mobile data processing system of claim **25** wherein the identifier data matches the second identifier and the location based action provides a location based feature intended for benefitting a data processing system user or group associated with the first identifier.

29. The user carried mobile data processing system of claim **24** wherein the first identifier grants the location based action to the second identifier.

30. The user carried mobile data processing system of claim **29** wherein the identifier data matches the second identifier and the location based action provides a location based feature intended for benefitting a data processing system user or group associated with the second identifier.

31. The user carried mobile data processing system of claim **29** wherein the identifier data matches the first identifier and the location based action provides a location based feature intended for benefitting a data processing system user or group associated with the second identifier.

32. The user carried mobile data processing system of claim **29** wherein the identifier data matches the second identifier and the location based action provides a location based feature intended for benefitting a data processing system user or group associated with the first identifier.

33. The user carried mobile data processing system of claim **24** wherein the first identifier grants the location based action to the first identifier, and wherein the identifier data matches the first identifier and the location based action provides a location based feature intended for benefitting a data processing system user or group associated with the first identifier.

34. The user carried mobile data processing system of claim **24** wherein the operations include:

accepting user input, from the user of the mobile data processing system, for administrating an other user specified location based event configuration including the first identifier and the second identifier, wherein the first identifier grants to the second identifier an other location based action to be performed by an other mobile data processing system, the second identifier being associated to the other mobile data processing system; and

communication the other user specified location based event configuration to the other mobile data processing system for effectual use at the other mobile data processing syste,

35. The user carried mobile data processing system of claim **34** wherein the first identifier is administrated for matching identifier data associated with a wireless data record processed by the other mobile data processing system and the other location based action provides a location based feature intended for benefitting a data processing system user or group associated with the second identifier.

36. The user carried mobile data processing system of claim **34** wherein the first identifier is administrated for matching identifier data associated with a wireless data record processed by the other mobile data processing system and the other location based action provides a location based feature intended for benefitting a data processing system user or group associated with the first identifier.

37. The user carried mobile data processing system of claim **34** wherein the second identifier is administrated for matching identifier data associated with a wireless data record processed by the other mobile data processing system and the other location based action provides a location based feature intended for benefitting a data processing system user or group associated with the first identifier.

38. The user carried mobile data processing system of claim 24 wherein the receiving for processing the wireless data record includes receiving for processing the wireless data record prior to transmitting the wireless data record outbound from the mobile data processing system.

39. The user carried mobile data processing system of claim 24 wherein the receiving for processing the wireless data record includes receiving for processing the wireless data record after inbound receipt by the mobile data processing system of the wireless data record.

40. The user carried mobile data processing system of claim 24 wherein the receiving for processing the wireless data record includes receiving for processing the wireless data record for insertion to a historical collection of information, the historical collection of information containing information for a plurality of data processing system detected by the mobile data processing system to have been in a wireless vicinity of the mobile data processing system.

41. The user carried mobile data processing system of claim 24 wherein the user specified location based event configuration includes a time or distance specification describing how the user specified location based event configuration is in effect.

42. The user carried mobile data processing system of claim 24 wherein the user specified location based event configuration can be cloned by a user of an other mobile data processing system for subsequent use by the user of the other mobile data processing system.

43. The user carried mobile data processing system of claim 24 wherein the first identifier or the second identifier or the third identifier or the identifier data identifies a group.

44. The user carried mobile data processing system of claim 24 wherein a user associated to a fourth identifier has an impersonation privilege for administrating the user specified location based event configuration.

45. The user carried mobile data processing system of claim 24 wherein the first identifier and the second identifier and the third identifier are each matched by the mobile data processing system to the identifier data.

46. The user carried mobile data processing system of claim 24 wherein the user specified location based event configuration grants at least one semantic privilege or at least one grammar specification privilege.

* * * * *