Step 1: Handle extensions to support event views

Initial VarHandle implementation depended directly on StoreGateSvc. But handles should work with event view objects too. Extend `IProxyDict` to include all operations needed by handles (eg, for recording). Then handles can depend only on `IProxyDict`, which can be implemented by views.

- Remove: `deep_proxy()`, `proxy(CLID)` (use variant with key instead). [Not actually removed yet from the interface, but there should be no callers.]
- Add:

```
SG::DataProxy*
recordObject (std::unique_ptr<DataObject> obj,
              const std::string& key,
              bool allowMods);

StatusCode updatedObject(CLID id, const std::string& key);
```

- Also: Merge functionality of `IProxyDictWithPool` into `IProxyDict`. [Still need to remove `IProxyDict`.]

Step 2: Hive fixes

Handles are problematic in hive: they cache a `DataProxy` pointer. But that depends on the event slot, which can change from call to call.

- Original version didn't clear the proxy, which is wrong. (Only worked because the only handle method that was actually used was a record method that didn't use the cached proxy.)
- As part of step 1, handle reset changed to clear proxy pointer.
- Caused problems: when reset is called from the event loop, the algorithm containing the handle instance may be processing a different event.
- In Hive mode, need to clear out handle at the end of the algorithm. But don't want to spoil caching in single-threaded jobs.

Step 2: Hive fixes

Solution:

- Distinguish 'hard' reset from a 'soft' reset by adding a boolean flag to `reset()`. On a hard reset, the event slot may be changing, so the proxy pointer needs to be cleared. A hard reset will be sent from SG if there's more than one active event slot.
- When a handle is bound to an object, the handle calls SG to put it on a list. SG then calls reset on this list when the store is cleared.
- In Hive, we were already calling at the end of each algorithm (via an auditor) `StoreGateSvc::commitNewDataObjects`. Extended this so reset all handles that were added since the last call to commit.
- The reset will then remove then handle from the store's list of handles.

Stateless handles?

Maybe we would be better off just making handles stateless?

- Does giving up the caching hurt performance?
- Probably not really, if we're just accessing the handle once per algorithm/event.
  - But don't want to do it within loops.
- Algorithms could fetch the pointer from the handle into a local at the start of execution.
  - But in that case, perhaps it's more natural to access it by calling an explicit method, rather than treating the handle as a pointer.
- But is it then so different from `retrieve()`?