Market Requirements Document

Feature Name: Objectivity/Mini

Version: 1 **Completed By:** Leon Guzenda **Date Submitted:** 03/02/07

Description of the Problem

True embedded databases must run in processors with minimal operating system functionality and with a footprint lower than 100KB. Objectivity/DB has a 1.5MB to 3MB memory footprint, plus cache.

Description of the Requested Feature

An embeddable, very low footprint, minimalist version of Objectivity/DB. It would support a single database. Appendix 1 covers the features to be provided.

Part of an existing feature or does it require another feature, if so, which one?

This feature is a tiny subset of an existing feature, but it will require extra build, quality assurance and documentation effort. Appendix 2 is a first attempt at a runtime model.

How is this problem being solved now, and why isn't that acceptable?

We do not offer a small footprint variant of Objectivity/DB. We have prospects requesting one.

What languages must support this capability?

- C++ first, then Java later.

Which platforms must be supported?

- Intel, AMD and ARM processors.

Do any competitors already have this feature?

- Most of the embeddable databases have footprints in the 200KB to 400KB (db4objects) range.

Customers who require this feature

- Western Digital.
- There is a potential market in the DoD, DoE, DHS and manufacturing industries.

Revenue at risk, or which could be won

- At 25 Cents per drive, the potential from Western Digital alone is around $5M per Quarter.

When is this required?

- Within three months.

Additional Notes

1. We will also need:

- Marketing collateral, including promotional material and a special area on our web site.

- Technical Publications.

- New QA material to prove that the reduced API works.

2. License enforcement will be a challenge. We should be able to include license checks in the development software, but the runtime profile may be too small for full license checking.

3. Licensing costs are to be determined. They should be realistic and should attempt to cover actual development and support costs.

Appendix 1 – Technical Requirements

1. The product will support a single database, equivalent to a single container, per runtime image.
2. The schema will be compiled into the runtime image.
3. Concurrency support (locking) will only be possible in environments that support shared memory.
4. Basic transaction support will cover start, commit and rollback functionality.

5. An in-memory transaction variant will rollback changes in memory but not affect the disk.
6. Page layouts will be identical to regular Objectivity/DB versions.
7. The embedded product need not support heterogeneous data, but the data it produces must be recognizable by other versions of Objectivity/DB.
8. It will be possible to create, open, update, close and delete objects.
9. Objects may consist of a fixed size body and at most one varying length data fragment.
10. Objects will have a 32-bit logical OID that can be converted to a regular OID by other versions of Objectivity/DB.
11. Object names and versions will not be supported.
12. Dynamic uni-directional and bi-directional relationships will be supported.
13. A navigation iterator will traverse relationships.
14. It will be possible to determine the class, superclass and subclasses of an object instance.
15. A scan iterator will find all instances of a given class or its subclasses.
16. Unsorted list and hash table (ooMap equivalent) collection classes will be supported.
17. An in-memory single or multiple key sort function will be supported.
18. Indexes will not be supported.
19. Iterators will support simple predicate queries.
20. The cache size may vary dynamically between lower and upper limits set at initialization time.
21. Objects may be cached across transactions.
22. The runtime code size cannot exceed 75KB.
23. An optional module could support an ODBC interface, but in a larger memory footprint.

Appendix 2 – Runtime Components

1. Transaction Manager
2. Cache Manager (Create, delete, expand and shrink memory spaces).
3. Persistent Storage Manager (create, delete, open, read, write persistent storage)
4. Schema Manager (compiled in structure definitions)
5. Object Manager (create, update, delete, open, close, add/remove link and iterators)
6. Collection Manager.
7. Predicate Manager (compiled in).
8. [Optional] ODBC server.