



White Paper

Birdstep RDM Embedded 6.0

A Technical Overview

July 25, 2002

Haakon VII's gate 5B, N-0161 Oslo, Norway
Tel: +47 24 13 47 00 Fax: +47 24 13 47 01
hello@birdstep.com www.birdstep.com

2101 4th Ave, Seattle WA 98121
Tel: 206 748 5353 Fax: 206 748 5200

Preface

This White Paper contains a technical overview of Birdstep RDM Embedded 6.0. This paper also lists the new features included in the RDM Embedded 6.0 release and summarizes a number of the improvements that have occurred to RDM since the 3.21a release. The intended audience of the paper is both the experienced RDM developer who is looking to determine the latest functionality of the database management system, as well those new to the RDM experience.

Copyright ©2002 Birdstep Technology Inc. Birdstep and RDM Embedded are a trademark of Birdstep Technology Inc. All other products mentioned bear the trademarks of their respective companies.

Table of Contents

BIRDSTEP RDM EMBEDDED 6.0 INTRODUCTION	4
NEW FEATURES	5
DATA REDUNDANCY	5
SQL INTERFACE	6
ENHANCED REENTRANCY	6
PLATFORM SUPPORT PACKAGES.....	7
ARCHITECTURAL UPDATES	ERROR! BOOKMARK NOT DEFINED.
LONGER FILE NAMES	8
CACHE ALLOCATION	8
LOCK MANAGER CHANGES	9
<i>lock manager Time Out</i>	9
<i>lock manager Service</i>	9
LOCK MANAGER FILE NAMING	10
UNICODE SUPPORT	11
<i>Unicode Data Fields</i>	12
<i>Sorting Unicode Fields</i>	12
<i>DDL and Unicode DBD Files</i>	12
DEBUGGING AIDES	13
PRODUCT COMPATIBILITY	15
LOCK MANAGER COMPATIBILITY	15
DATABASE COMPATIBILITY	16
APPLICATION COMPATIBILITY	16

Birdstep RDM Embedded 6.0 Introduction

Over its 17-year history, Birdstep RDM Embedded has always been driven by the needs of our developer community, as such, features and changes have been included over the years to address their needs. As operating systems have advanced, many of the features and improvements that were once important are no longer necessary. To continue to address the needs of our developers, Birdstep RDM Embedded 6.0 is a complete rework of the RDM database engine. Every line of every source file was examined to determine why it was there, if it was needed, and could it be done in a better way. The result is an engine that is faster, more robust, and better equipped to handle the requirements of the modern embedded developer.

New Features

Data Redundancy

New for Birdstep RDM Embedded 6.0 is a feature that adds data redundancy services to the database engine. Many embedded operating systems lack a disk based mirroring solution. In this case, an engine level solution creating a near real-time backup of the production database to be created is appropriate.

The RDM Embedded mirroring system is designed to provide an integrated, engine-level solution for mirroring main database modifications to a single mirror copy of that database. The storage media for both the main and mirrored data is not important, providing the media has a file system interface. For, example the main database may be stored in volatile main memory, while the mirror is stored in persistent flash memory. The main database will have the speed of direct memory access coupled with the ability to persistently store data. Another example may have the main and mirror stored on separate physical hard drives. The mirror database will provide rollover capability if the main database becomes corrupt, or if there is a failure on the main database drive.

The mirroring system will assist developers in creating Fault Tolerant applications; however, the design does not attempt to make fault tolerance an engine feature. It will be up to the developer to utilize the data redundancy features provided by the engine to implement the specific level of fault tolerance required within their application suites. If the main database becomes unavailable it is up to the developer to utilize the mirroring system features to refresh the main database, or to begin accessing the mirror.a

There are two major components comprising the mirroring system. The first component mimics the transaction logging subsystem of the RDM Embedded runtime library, to create mirror log files containing binary copies of all the modifications occurring to the main database during a single transaction. The second component takes the mirror log files and applies the stored modifications to the mirror copy of the database. This can be done either synchronously or asynchronously. To provide as little performance impact as possible it is suggested to use the asynchronous mirroring model.

An important part of the synchronization framework is the implementation of communication and control between the two mirroring components. The synchronization and communication mechanisms between are based on OS specific inter-process and intra-process communication facilities: message queues, mutexes, semaphores, and events. The initial implementation of this data redundancy system requires both the RDM Embedded application and the mirror application component to be running on the same computer/device. It is possible to use mapped or NFS mounted drives to mirror

to a drive hosted on a remote machine, but currently the application and mirroring system only run together.

SQL Interface

New for RDM Embedded 6.0 is a replacement for dbQuery in the form of a complete SQL interface. The new SQL interface includes the ability to query and update RDM databases using industry standard SQL statements presented to the engine via an ODBC 3.5 like interface.

The API's listed below are a subset of the ODBC 3.51 standard, chosen to best reflect the needs of the RTOS market.

API	ODBC 3.51	RDM Embedded SQL Interface
SQLBindCol	C	v
SQLConnect	C	v
SQLDescribeCol	C	v
SQLDisconnect	C	v
SQLExecDirect	C	v
SQLExecute	C	v
SQLFetch	C	v
SQLGetCursorName	C	v
SQLNumResultCols	C	v
SQLPrepare	C	v
SQLRowCount	C	v
SQLSetCursorName	C	v
SQLNumParams	C	v
SQLBindParameter	C	v
SQLAllocHandle	C	v
SQLCloseCursor	C	v
SQLEndTran	C	v
SQLFreeHandle	C	v
SQLGetDiagField	C	v
SQLGetDiagRec	C	v
SQLSetConnectAttr	C	v
SQLDescribeParam	2	v
SQLSetStmtAttr	C	v

Enhanced Reentrancy

Our first fully reentrant release of RDM Embedded on all supported platforms was version 5.0. Although RDM 4.5 supported multi-threaded applications, the runtime library only allowed a single thread to execute its code at a time - if multiple threads called its functions simultaneously then all, except one of them, would be blocked and they would execute the RDM code serially. In RDM Embedded 5.0, the runtime library allows simultaneous execution by multiple threads. This can lead to significant performance improvements in a multi-threaded application.

If your application has multiple threads that call RDM or Raima Object Manager (ROM), you must open a separate database task for each thread (with `dt_opentask`). Sharing a database task between threads will cause memory corruption and possibly database corruption.

Birdstep RDM Embedded 6.0 enhances the reentrancy features of RDM. The re-architecting of RDM Embedded 6.0 was done with reentrancy in mind from the ground up. To support enhanced reentrancy we have changed the API set. No longer are there two versions of every function (`d_` for single-tasking, `dt_` for multi-tasking). Now there is only a single set of `d_` APIs allowing for both sets of functionality. For developers who do not want to modify their source to utilize the new API structure, RDM Embedded 6.0 provides a rich set of mappings allowing use of the new features without changing a single line of your code.

Platform Support Packages

Birdstep RDM Embedded 6.0 has been designed to include all platform specific code into a single set of modules. The core engine uses the services of the Platform Support Packages (PSP) instead of directly calling platform specific services. Developers using RDM Embedded will benefit greatly by our platform specific code.

Developers with an application supporting multiple operating systems, using our PSP services can greatly reduce the effort required to develop and maintain your application.

Standard Features

The following sections list some of the architectural changes that have been made to RDM Embedded since the release of version 3.21a.

Longer File Names

Versions of RDM Embedded, previous to 4.5, imposed a limit on the length of all database filenames of 47 characters; including path. This limit has been changed to 255 characters in RDM Embedded 4.5 and beyond.

The database format (including the dictionary file, .dbd) has not been changed. Because the DBD file contains the names of all data and key files in the database, as specified in the DDL schema, there is still a limit of 47 characters on the names in the DDL schema (and the DBD file). Since it is not recommended to hard-wire paths into the database definition, we hope this will not prove restrictive.

The path of database files can be specified in several ways, such as in the `d_open()` call, through `d_dbdpath()` or `d_dbfpath()`, or through the `rdm.ini` file. All of these methods allow a 255-character path.

Note that if you have written any programs that read the RDM database dictionary directly, and you are using the `FILE_ENTRY` structure defined in `dbtype.h`, you will need to change your programs so that they use `V3_FILE_ENTRY` instead.

Cache Allocation

RDM Embedded uses file locking to ensure data reliability in a multi-user environment. RDM also uses a cache to increase performance, so that a page that was read recently is reread from memory rather than from disk when it needs to be accessed, which saves a great deal of time. The interaction of locking and caching in previous versions was simple: when a lock was obtained, all cache data for that file was discarded.

With the last release of RDM Embedded, version 5.0, a new strategy was adopted. For some users, it significantly enhanced performance, and without reducing reliability. For each file handled by the lock manager, a "timestamp" value is retained. Each time the file might be written to (either a write or an

exclusive lock is obtained) the timestamp is incremented. The runtime is sent this value, and only flushes its cache when the timestamps do not match.

For applications that have a lot of data in cache, can effectively reuse it, and practice good lock discipline by releasing locks often, this can dramatically increase cache effectiveness when not a lot of writing occurs.

Lock Manager Changes

The lock manager is among the most intricate pieces in the RDM Embedded system. As a result many of the RDM deficiencies reported to Birdstep Technology relate to the lock manager. Since the release of RDM Embedded 3.21a there have been hundreds of updates and revisions to the lock manager code line. In fact, we have retired multiple lock managers completely. The following lock managers are supported with RDM Embedded 6.0:

- **Win32 Systems**
 - TCP/IP
 - Internal
- **Unix Like Systems**
 - TCP/IP
 - Internal (*Not available on Linux*)
 - Domain Socket (*Not available on QNX Neutrino*)

Each of these lock managers is significantly more stable than any lock manager available for previous RDM Embedded releases. The following changes have been made to the lock managers.

lock manager Time Out

This feature regulates concurrent access to data in a multi-user environment, resulting in freeing data for other users. This makes RDM Embedded a self-managing database, therefore eliminating the need for human intervention to remove a failed client.

- New API: `d_locktimeout`: Takes 2 parameters, read lock timeout and write lock timeout.
- Very aggressive API and should only be used when there's no other methods of kicking out an idle user.
- If a read or write lock is held too long, the lock manager will kill that user.
- Exclusive locks will never timeout.
- Could cause data loss in case a write lock is released. For this reason, this API should be used with caution.

Lock Manager Service

The lock manager can now be run as a service. In order to take advantage of this feature, it is necessary to install and configure the Win NT files `instsrv.exe` and `svany.exe`. Please refer to the RDM as a service section of `readme.txt` for more info and setup procedures.

The lock manager has a new command line option `-v`. This option when turned on hides the user interface of LMW, allowing it to run in the background.

If you experience problems with the lockmanager as a service, try running it without the -v option. This will allow you to see the errors. (When -v is turned on, you will not see any error codes and nothing will be written to the event viewer.)

Lock Manager File Naming

The lock manager works by maintaining a list of files that are used by the RDM Embedded applications communicating with it. When an application opens a database it sends a list of the names of all the database files to the lock manager. Therefore it is necessary, that all applications use the same names to identify these files. If different applications use different names for the same file then the lock manager will treat these as different files, and may grant simultaneous write locks to multiple applications on what is in fact the same file. This will probably result in database corruption.

When sending a list of filenames to the lock manager the RDM Embedded runtime library refers to database files by their UNC names (Universal Naming Convention), to avoid inconsistencies caused by different drive letter mappings on different workstations. UNC names consist of a machine name followed by a path, in the form: [\\machine_name\directory\filename](#)

The use of UNC filenames is enabled through the RDM Embedded runtime option LMC_OPT_TRUENAME, which is on by default. All versions of RDM Embedded, since 3.30, have used this mechanism.

Use LMC_OPT_NOPATH setting

A runtime option has been added which removes the path from all filenames sent to the lock manager. For example, the file

C:\RDMDB\TEST\TESTDB.D01

would become:

TESTDB.D01

This will solve the problem provided the filenames managed by the lock manager are unique. However, if you have multiple instances of the same database in different directories, and these are opened in shared ("s") or exclusive ("x") mode, this solution will not work. Also, this option is not supported by any existing 32-bit version of RDM Embedded.

To enable this option, either add the following entries to the rdm.ini file:

```
[rdm]
truename=0
nopath=1
```

or else, call the following functions before calling dt_open():

```
dt_off_opt(LMC_OPT_TRUENAME, &Task);
dt_on_opt(LMC_OPT_NOPATH, &Task);
```

TRUENAME CALLBACK FUNCTION

The application can supply a callback function which performs customized translation of filenames to UNC names. RDM Embedded will then call this function whenever it needs to convert a file name to UNC format, for dispatch to the lock manager (when a database is opened). The callback function will be called once, for each file- name that needs to be converted, with the filename passed as an argument. It may convert the file name to UNC format, or it may return without action; its return value should indicate which of these it has done.

To use a callback function you will need to do the following:

- Write the callback function itself
- Add the callback function to the list of exports from the application, in the DEF file (Windows applications only)
- Make sure that the RDM Embedded runtime option LMC_OPT_TRUE_NAME is enabled (which is the default setting)
- Call MakeProclInstance() to obtain a pointer to the function (Windows applications only)
- Pass the pointer returned by MakeProclInstance() (Windows) or the address of the function itself (MS-DSO) to the RDM function dt_set_dbtruenam() before calling dt_open()
- Test that your callback function is working by examining the filenames in the lock manager's file list - to see the full path of a file, select the file and click on the "View File" button (Windows lock manager), or type "FILE n" where n is the file number (MS-DOS lock manager)

Unicode Support

RDM Embedded 6.0 provides two levels of support for Unicode (on platforms that support Unicode):

1. Unicode data fields (data type wchar_t)
2. Unicode library functions, with Unicode string arguments and full internal string handling in Unicode

The RDM binaries provided with this product consist of two versions of each program and library: an 'ANSI' version, that supports Unicode data fields, but takes ANSI string arguments, and does all internal string handling in ANSI strings (level 1 above), and a 'Unicode' version that is fully implemented in Unicode (levels 1 and 2 above).

The functionality of the two sets of binaries is identical, except for Unicode support. There is no advantage in using the Unicode DLL in an application that does not manipulate foreign language data. In fact, since Windows 95 and Windows 98 do not fully support Unicode, we do not recommend using the Unicode binaries unless you must. If you are developing a new RDM Embedded application, then keep in mind that use of Unicode will effectively constrain its deployment to Windows NT and Windows CE.

If you use the Unicode DLL then we recommended that you always use the Unicode utilities as well. Similarly, if you use the ANSI DLL, then you should always use the ANSI utilities. This is because the Transaction Activity Files (TAF files) generated by the ANSI and Unicode DLLs are not compatible. If you mix the ANSI and Unicode utilities you will find you continually get S_TAFSYNC (-944) errors, and have to delete the TAF file whenever this happens.

Unicode Data Fields

Both the ANSI and Unicode versions of ddp recognize the data type `wchar_t` for database fields. In the database dictionary (DBD file) these fields are represented by the letter 'C', similar to 'c' for char fields. The size of these fields is system dependent - on Win32 a `wchar_t` is two bytes in size, and on UNIX it is 4 bytes. Unfortunately, this leads to an incompatibility between data on different platforms, but differences in byte ordering would often cause incompatibilities anyway.

Sorting Unicode Fields

RDM Embedded uses the C runtime library functions `_wcsncoll()` and `_wcsnicoll()` to sort Unicode strings that are defined as key fields or used in sorted sets. If RDM Embedded's IGNORECASE option is enabled, `_wcsnicoll()` will be used, otherwise `_wcsncoll()`.

These functions use the selected locale to determine how strings should be collated. The C runtime library selects the 'C' locale on startup, resulting in the ASCII sorting order. In general, however, the ASCII sorting order is not suitable for foreign language data, so RDM Embedded selects the system default locale on initialization, restoring the original locale setting on termination (Specifically, the locale is set when a process attaches to the RDM Embedded DLL, and reset when it detaches from the RDM Embedded DLL - see `libmain.c` and `global.c`).

You can override RDM Embedded's locale setting by calling `setlocale()` or `_wsetlocale()` in your application. If you require case-sensitive sorting, as in the ASCII sorting order, you should select the 'C' locale.

Note that the RDM Embedded Country Table is not used in sorting Unicode data. This feature, which has been supported in all versions of RDM Embedded since 3.0, allows the application to redefine the sorting order of char data. It is used in European language applications, where characters are not sorted by their ASCII values. Since these characters are sorted correctly by the Unicode string collation functions anyway, there is no need for any mechanism, such as the Country Table, in sorting Unicode strings.

The Country Table is, however, still usable with char fields.

ddlp and Unicode DBD Files

The Unicode version of ddp (`ddpu.exe`) normally requires the DDL schema to be an ASCII text file, but if the '-u' option is specified on the command line then the DDL schema must contain Unicode text.

Likewise, the database header file generated by ddp will normally be an ASCII text file, but if the '-u' option is specified then it will be in Unicode text (Note that Microsoft Visual C++ will compile Unicode text files). The database filenames in the database dictionary will also be represented in Unicode text if the '-u' option is used. The DBD file generated will be in u5.00 format, which is different from the v3.00 format used in prior versions of RDM Embedded. The RDM Embedded 5.0 DLL can read DBD files in either format, but the new format is not readable by older versions of RDM Embedded.

Debugging Aides

Starting with version 4.0, RDM Embedded contains several debugging switches to assist RDM Embedded developers in isolating any problems that could be related to the runtime library. They are enabled when the library or DLL is re-compiled with the DB_DEBUG define.

When DB_DEBUG is defined, the following runtime options become available by calling d_on_opt() or by adding entries in the rdm.ini file in the [debug] section:

Option	Description
PZVERIFY	Upon access to a database file, next_slot and delete_chain values are compared against the actual file size to ensure that they do not point past the end of the file. This will help detect an exploding file size problem.
PAGE_CHECK	Each database page read into the cache will be marked with an identifying number so that it can be tracked in memory and validated upon write. This provides only minimal verification, and is not expected to be of much use for a customer.
LOCK_CHECK	Upon each file-lock protected access to a database file, or page in the cache, the last modified date of the file is retrieved, and the last modified time in the first 4 bytes of the page are read from the disk to ensure that another process has not illegally updated the file since it was locked. This can detect improper lock arbitration due to misnaming of the TAF file, lock manager, or database filenames. This will detect the common problem of the lock manager seeming to have duplicate file names when not all processes are using the TRUENAME option in the same way. !!! WARNING !!! This option performs disk I/O on each access to the internal cache and will greatly decrease your application's performance.
CACHE_CHECK	The internal cache data structures are checked for consistency at the beginning of each d_ function, and after each cache reference.

In addition, DB_DEBUG also defines DB_TRACE, which enables the runtime library to write various trace output to a disk file. The name of the file is hard-coded to be "vista.out", but its location can be changed by setting a new environment variable, DBTPATH. The trace file has been designed to be shared by multiple processes on the network - it is arbitrated between processes by the use of share locks, and each time a different DB_TASK writes to it, a line identifying the process and task is also written.

- TRACE_DBERR** If this, or any other trace option is on, all database errors will be logged to the trace file immediately prior to calling your dberr handler.
- TRACE_API** Each d_ function will be written. Since the rdm.ini file is not processed until the first call to d_open, any pre-open d_ function will not be traced unless the DBTRACE environment variable is defined. If defined, it will turn on all trace options until overridden by the rdm.ini file.
- TRACE_LOCKS** An entry will be written each time a file is locked or freed.
- Note:** The trace file will be physically committed to the disk on each write if the SYNCFILES option is turned on (the default). This prevents any output from being lost if your program terminates with a GP fault or a hung machine, but causes a tremendous amount of disk overhead.

A new section has been added to the rdm.ini file for specifying the debug and trace options:

```
[debug]
pzverify=1           ; verify pgzero against filelength on each access
page_check=1        ; verify cache page identity
lock_check=1         ; detect unarbitrated file updates
cache_check=1 ; verify cache consistency
trace_dberr=1        ; output dberr messages to trace file
trace_api=1          ; write all d_ api functions to trace file
trace_locks=1        ; write file locks and frees to trace file
```

These options can also be set with d_on_opt() and d_off_opt(), however, the debug options and the trace options cannot be mixed with each other or with any other options in the same d_on_opt() or d_off_opt() call.

```
d_on_opt(PZVERIFY);
d_on_opt(TRACE_DBERR | TRACE_API); /* must be separate call */
```

Two environment variables have been added for controlling trace output:

```
set DBTPATH=<path> ; set location of the "vista.out" trace file
set DBTRACE=1      ; enables full tracing on program startup
```

Finally, note that if the library or DLL is not compiled with the DB_DEBUG or DB_TRACE defines, this functionality will be omitted from the library, and any references to these options will be ignored.

Product Compatibility

- The general lock manager has been dropped.
- ROM (Raima Object Manager) is no longer a part of the product and is no longer supported.
- With the addition of RDM Embedded SQL, there is no longer a need for db_QUERY. Thus db_QUERY is no longer a part of the product. For customers who utilize db_QUERY in their applications the source for db_QUERY has been included with the distribution, but it is no longer a supported feature. We suggest that you work to migrate your db_QUERY modules to utilize the RDM Embedded SQL features.

Operating Systems & Compilers Supported

- AIX 4.3 with C Set ++ for AIX
- HP-UX 11 with ANSI cc compiler
- Microsoft Windows XP, NT, 95/98/Me with Microsoft Visual C++ (Compiler version 12.x)
- Microsoft WinCE 3.1 with Microsoft Embedded Visual C++3.0
- QNX RTOS v6 (Neutrino) with gcc version 2.95.2
- Red Hat Linux 7.1 & 7.2 with gcc version 2.96
- Solaris (Sparc & x86) 5.5 & 5.6 with WorkShop Compilers 4.2
- Solaris (Sparc & x86) 5.7 & 5.8 with WorkShop Compilers 5.0
- Unixware 7.0.1 with Optimizing C Compilation System (CCS) 3.2
- Windriver VxWorks 5.4 with Tornado 2
- Windriver VxWorks AE with Tornado 3
- Compatible with LynxOs, .Net and Nucleus
- Other operating systems: [Contact us](#) for a quote.

lock manager Compatibility

It is not possible to run RDM Embedded 6.0 and earlier versions of RDM Embedded simultaneously, sharing databases, in the same a multi-user system. The data structures listed below are incompatible between RDM Embedded 6.0 and prior versions.

- Incompatible data structures
- lock manager messages
- Transaction Activity File (TAF)
- Transaction Log Files
- Database Lock (DBL) File

If you are upgrading from an earlier version of RDM, please delete all TAF, LOG and DBL files.

Note that the above structures are also incompatible between the Unicode and ANSI versions of RDM. Therefore, it is impossible to run Unicode and ANSI RDM applications simultaneously in the same multi-user system.

Database Compatibility

Databases created with earlier versions of RDM Embedded are compatible with RDM Embedded 6.0. The format of data and key files has not changed in RDM Embedded 6.0 and, although a new Database Dictionary (DBD) format has been introduced, RDM Embedded 6.0 still recognizes the old format and can still generate DBD files in the old format.

RDM Embedded 6.0 supports the following DBD formats:

- v3.00 Used by all versions of RDM Embedded from 3.0 to 4.5, with ANSI data and key filenames up 48 characters long
- v3.01 Previously only recognized in RDM/Vx 4.9.3, with VxWorks-specific DDL extensions
- v5.00 ANSI filenames up to 256 characters long, and v3.01 extensions
- u5.00 Unicode filenames up to 256 characters long, and v3.01 extensions

These DBD versions are supported by the RDM versions shown in the table below:

	RDM Embedded 3.0 to 4.5	RDM Embedded 4.9.3	RDM Embedded 5/6 Ansi	RDM Embedded 5/6 Unicode
V3.00	Yes	Yes	Yes	Yes
V3.01	No	Yes	Yes	Yes
V5.00	No	No	Yes	Yes
U5.00	No	No	No	Yes

Application Compatibility

The RDM Embedded 6.0 API is slightly different than the two API's provided by RDM Embedded 5.0 and previous versions of RDM, thus this release provides optional translations from the previous API to the new form of the RDM Embedded 6.0 API. Programmers may choose to leave existing applications unchanged by using one of the provided translations, or they may change their source to the new API. To utilize the translation, preprocessor constants must be defined, either on the command line or in application source code prior to the RDM Embedded 6.0 headers.

Applications utilizing translations must include the header vista.h, as in versions previous to RDM Embedded 6.0. To translate from RDM Embedded 5.0's dt_api to the new form RDM Embedded 6.0 define the preprocessor symbol RDM_LEGACY_API. To translate from RDM Embedded 5.0's d_api without a database task parameter define RDM_TASK in addition to RDM_LEGACY_API.