Polyhedra®

Polyhedra 9.0 Release Information

Enea Polyhedra Ltd

ENEA

Copyright notice

Disclaimer

Trademarks

Manual Details

Manual title	**Polyhedra 9.0 Release Information**
Document number	**src/release/readme**
Revision number	**9.0.19**
Revision date	**15th June 2015**

Software Version

This documentation corresponds to the following version of the Polyhedra software:

Version	**9.0**

Preface

This document contains release information for Polyhedra version 9.0.

CONTENTS

1. Introduction

This document contains release information for Polyhedra 9.0 and is targeted at existing users of Polyhedra who are considering migrating their applications to take advantage of the functionality enhancements of this new release. It describes:

- changes
- migration
- corrections
- known problems

2. Installation

The installation procedure for Polyhedra 9.0 is described in *Installing Polyhedra*, included as part of each release kit and also available separately via our support web site.

3. Changes

The following changes have been implemented in Polyhedra 9.0. The previous release was Polyhedra 8.9.

3.1 New Platforms Supported in 9.0

No new platforms have been introduced in this release.

3.2 Platforms Not Supported in 9.0

The following platform is not supported in this release:

- Polyhedra for OSE on Solaris SFK using GNU C/C++

3.3 Updated Platforms Supported in 9.0

The supported operating system for the following release kits is now OSE 5.8:

- Polyhedra for OSE on PowerPC using GNU C/C++
- Polyhedra for OSE on Linux SFK using GNU C/C++
- Polyhedra Flash DBMS for OSE on PowerPC using GNU C/C++

3.4 New Software Components in 9.0

The following new software component has been introduced in this release:

- Polyhedra ADO.NET Data Provider (32-bit and 64-bit)

3.5 New Features and Improvements in 9.0

The following new features have been introduced in this release:

3.5.1 Data Subscription

Polyhedra now supports a facility where one RTRDB can subscribe to a subset of tables in another RTRDB. Changes to data in the source tables are applied to the corresponding tables in the destination. This new facility is in contrast to the existing data replication feature where the entire database must be replicated.

This new feature in described in the Polyhedra *Real-Time Relational Database* manual.

3.5.2 ADO.NET Data Provider

A new Polyhedra ADO.NET Data Provider for use with the Microsoft .NET Framework is included in this release. It allows a .NET application to access data stored in a Polyhedra database. In addition to the standard features of an ADO.NET data provider it also supports Polyhedra specific features such as fault tolerance and active queries.

This new component is described in the *Polyhedra ADO.NET Data Provider* manual.

3.5.3 Callback API

The following enhancements have been made to the Callback API:

- A new **CommandAPI** class has been introduced to allow separate preparation and execution of SQL statements. Using the new class SQL statements can be prepared once and then executed multiple times with different

parameter values. They can be executed either as fixed or active queries or as part of a transaction. The new **CommandAPI** class is derived from the existing **QueryAPI** class.

- A new **TransAPI::AddCommand** function has been introduced to allow a command, whether prepared or not, to be added to a transaction so that it is executed when the transaction is committed. A transaction may therefore now contain a combination of updates through active queries and SQL statements.
- A new **TransAPI::GetSQLRowCount** function has been introduced that returns the number of records directly modified by SQL statements in a transaction.
- A new **QueryAPI::GetColumnFlags** function has been introduced to obtain the flags associated with a column returned by a query. The flags indicate whether the column is a primary key column, can contain null values and can be updated.
- The existing **QueryAPI::AddArg** function has been extended to accept a null name argument to be passed to indicate the parameter has no name. It is now possible to reference unnamed parameters using positional parameters markers ('?') in SQL text.
- The existing **QueryAPI::AddArg** function has also been extended to allow a null value argument to be passed.
- The existing **QueryAPI::GetError** function has been extended to allow the underlying database error code and text associated with an error to be obtained.
- The existing **QueryAPI::StartAbort** function does not delete the aborted active query if the query is an instance of the new **CommandAPI** class.
- The existing **ClientAPI::GetOption** function supports a new option value POLY_OPTION_DBMS_VERSION that returns the version of the RTRDB to which the client is connected encoded as an integer.

All new and extended Callback API functions have C language equivalents and are accessible via the Callback API DLL on Windows.

These enhancements are described in the *Callback API* manual

3.5.4 Embedded Interface

The following new user-supplied functions have been added to the Polyhedra Embedded Interface:

```
void poly_thread_entry();
```

This function is called whenever a Polyhedra component creates a new thread of execution.

```
void poly_thread_exit();
```

This function is called whenever a thread executing in a Polyhedra component terminates.

These new functions are described in the *Polyhedra on Embedded Systems* manual.

4. Migrating to Polyhedra 9.0

This section details how to migrate an application from an earlier version of Polyhedra to Polyhedra 9.0.

Except as described below, Polyhedra 9.0 does not introduce any incompatibility with and is interoperable with Polyhedra 8.9. Polyhedra 9.0 will act as a maintenance release for 8.9. There will be no further 8.9.X maintenance releases.

4.1 Callback API

For consistency with other Callback API functions that obtain error information the prototype of the function **QueryAPI::GetError** and its C language equivalent function **QueryAPI_GetError** have been extended with additional parameters. Existing client applications that call **QueryAPI::GetError** will not need to be modified as the new parameters have suitable default values, but they will need to be re-compiled before linking with the new Callback API library. Existing client applications that call **QueryAPI_GetError** will need to be modified to pass values for the additional parameters when upgrading to use the new Callback API library.

The new **CommandAPI** class allows parameterised SQL statements to be prepared and then executed multiple times with different parameter values. A prepared statement is more efficient to execute than one that has not been prepared as the compilation and optimisation of the statement is performed once when it is prepared.

Whenever possible when executing transactions it is advisable to re-use instances of the **TransAPI** class as deleting an instance (using the **TransAPI::DeleteTrans** function incurs the overhead of a client-server message round-trip.

Using the newly supported positional (unnamed) parameters is more efficient than using named parameters as they do not require a name look-up in the RTRDB.

4.2 Embedded Interface

Existing applications that use the Polyhedra Embedded Interface will need to provide implementations of the new **poly_thread_entry** and **poly_thread_exit** functions.

4.3 SQL Identifiers

The RTRDB now accepts identifiers that contain consecutive underscores and/or trailing underscores. This is a relaxation of previous rules.

4.4 Ordered Indexes

A composite ordered index will now not be used during query evaluation if any unmatched columns in the index allow NULL values.

4.5 OSE Release Kits

The OSE development release kits for PowerPC, Linux Soft Kernel and Solaris Soft Kernel now include object and library files for linking the Device Interface (DVI) module into the RTRDB software component. Previously these had been omitted. Note that DVI module is optional and so existing commands for linking the RTRDB will be unaffected.

4.6 Domains

The choice of which value is assigned to a shared or virtual column in a domain where the domain type is present more than once in a table is undefined for inserts and updates. The fix for product defect 11895 has affected this choice. Therefore it is recommended that such a statement should only set a shared or virtual column in a domain once.

4.7 Building from Source

To build Polyhedra from source now requires GNU Bison version 2.4.1 or later.

4.8 Replicas

The default value for the resource **journal_connect_interval** has been changed from 5000 (5 seconds) to 1000 (1 second) to be consistent with the default connect interval used in table subscription. The previous behaviour can be restored by adding a **journal_connect_interval** resource of 5000.

4.9 CL Bell Class and Beep Command

The CL bell class and beep command have been deprecated.

5. Corrections

This section details the corrections that have been made to Polyhedra 9.0.

5.1 Full Release: 9.0.0

All bugs fixed in Polyhedra 8.9 up to the most recent maintenance release 8.9.1 have been checked against Polyhedra 9.0 Release and, where appropriate, are also fixed in this release. The following bugs present in 8.9.1 and earlier are fixed by this release:

- 6187
  The Callback API function TransAPI::DeleteTrans could not be called if the connection had been lost. This could result in a memory leak.
- 7190
  On Windows updating a character column when using UTF-8 character set could, in rare circumstances, fail silently.
- 7506
  Polyhedra development release kits for OSE now include object and library files for linking the DVI module into the RTRDB.
- 10576
  A client application could crash, or crash the RTRDB, when it issued a transaction containing a large number of named parameter values.
- 11485
  An SQL identifier may now contain consecutive underscores and may end with an underscore.
- 11546
  The RTRDB did not prevent the DATACONNECTION table being altered to be local when the DATAPORT table was non-local.
- 11686
  On rare occasions in a fault tolerant setup starting up a replica could cause the standby to terminate.
- 11737
  A join on an ordered index varchar column could crash a Flash DBMS RTRDB.
- 11765
  The default data service connection used by CL in the RTRDB now has SYSTEM access if security is enabled.
- 11767
  A Flash DBMS RTRDB could crash or hang if a record was deleted while being updated through an active query.
- 11768
  The merge utility could fail to merge historian archives of a compressed stream if a raw LOGCOLUMN of the stream had INDEXMETHOD set.
- 11770
  The CL beep command, which is now deprecated, would crash the CLC.
- 11794
  On rare occasion an RTRDB that was restarted from log files produced on a standby could produce archives which contained blocks that had already been archived previously on the master.
- 11803
  A unique or non-unique user hash index on a foreign key column (not a primary key) could fail to reload correctly after a warm restart. This would cause queries that used the index not returning all the correct records.
- 11806
  The RTRDB would incorrectly load a corrupt database load file that contained duplicate records.

- 11828
  A performance issue with SQL foreign key referential actions has been addressed.
- 11834
  The CL Bell class, which is now deprecated, would crash the CLC.
- 11838
  The echo_commands=true resource now outputs prompt and transaction completed text.
- 11839
  A < or > comparison of a VARCHAR(n) in a join could give an incorrect result in Flash DBMS.
- 11841
  A query that used only some columns of a composite ordered index could return the incorrect rows if the unused columns contained NULL values.
- 11845
  Fix a potential hang that could occur when a latest version RTRDB is being used as a standby to a version 8.5 RTRDB master. (The RTRDB must be using the historian component).
- 11847
  Programme output was seen when a duplicate key was detected in a unique ordered index.
- 11858
  A small amount of memory may not have been freed when a CLC that was using active queries terminated.
- 11859
  The RTRDB would crash when doing a warm restart if the JOURNALPORT or DATAPORT tables were in use.
- 11864
  The RTRDB could crash on Windows when comparing large UNICODE strings.
- 11871
  It is now possible to set the default value of the Callback API option POLY_OPTION_CONNECTION_TIMEOUT.
- 11872
  Restore the ability to nest calls to the Callback API function AppAPI::Execute() on OSE.
- 11890
  Using the repeat command (/) in SQLC did not always work if the last command was the "reading" command.
- 11895
  The RTRDB could crash when inserting a record into a domain that was derived from another domain and also included a column with the same type as the domain which it derived from.
- 11898
  DateTime values for the year 1753 were handled incorrectly - they could be out by 11 days. This problem was confined to DateTime values in this year.
- 11904
  The value returned by the ODBC function SQLRowCount() could be incorrect for non-DML statements. The value returned for non-DML statements should be 0, but the row count for the previously executed SQL DML statement was returned instead.
- 11910
  The RTRDB could give incorrect results on Windows when comparing UNICODE strings.
- 11934
  Changing permissions on a table that is used by a large number of views which reference tables containing a large number of records could take a longer time than necessary.
- 11963
  A comparison between a binary containing a null character and a constant binary value would be incorrect on the Flash DBMS.
- 11950
  The animate sample ODBC client could crash when more than one instance of it was run at the same time against the same database.
- 11964
  An active query would stop generating deltas after it had encountered a data conflict error.

6. Known Problems

The following known problems exist in this release:

- 5337
  The OLE DB Provider can occasionally crash when attempting to do an update on a row if that row had just been deleted in the RTRDB.
- 5840
  The client library re-executing an active query after fail-over assumes that the structure of the result set is compatible with the original execution of the query.
- 6483
  On most platforms the maximum number of sockets is compiled into the RTRDB. If running on a system where the maximum number of sockets has been reconfigured, the RTRDB can give undefined results.
- 6631
  The OLE DB Provider incorrectly includes a provider-specific property with description "FT Keep Copy" that is visible to consumers. If set to false, a fault-tolerant connection might disconnect at failover time.
- 6653
  The RTRDB can take a long time to shutdown when using a memory-mapped file on the Win32 platform.
- 6672
  The debugger does not generate trace information for all transactions.
- 6690
  Using the ODBC API if the RTRDB is killed before a call to SQLPrepare when asynchronous execution is selected, then SQLPrepare will never stop returning SQL_STILL_EXECUTING.
- 6833
  A fault tolerant standby incorrectly creates log files that have been abandoned by the master.
- 7323
  The ISPRIMARY field of the attributes table can be set to true incorrectly for a column in view that is used in a join restriction.
- 7402
  Using the ODBC API it is not possible to use the combination of manual commit and asynchronous execution.
- 7592
  Updates to historical data are not journalled if the transaction is from an ODBC client using SQLExecDirect or SQLExecute in auto-commit mode.
- 8971
  On OSE when using the CL File class, the Write function returns true even if no characters are written because the Mode is "read".
- 9202
  Using the Flash DBMS RTRDB any persistent shared or virtual columns in a transient table will have null values when the database is restarted.
- 9250
  Enabling a previously disabled DataPort using non-TCP transport may cause a resource leak.
- 9291
  There is a discrepancy between Polyhedra IMDB and Flash DBMS. IMDB and Flash DBMS have different behaviours regarding string and binary foreign keys: standard ignores the length constraint of the foreign key whereas Flash DBMS takes notice, truncating before matching.
- 9311
  Defining a stored procedure that deletes itself will crash the RTRDB.
- 9314
  Executing a procedure from the Callback API bypasses the security system.

- 9325
  The value of the client_type column of the dataconnection table is not set for client connections to a master RTRDB that existed before it was promoted from being a standby.
- 9332
  Heartbeat messages are still sent to the TCP arbitrator by the RTRDB when the heartbeat interval is 0.
- 9408
  The RTRDB incorrectly allows a persistent foreign key reference to a transient table.
- 9833
  The RTRDB incorrectly accepts a schema definition containing a persistent foreign key reference to transient data.
- 9901
  SQLC and Callback API clients will hang if an undefined transport is specified in the data service.
- 9905
  Resource recovery does not always cope with failure to allocate memory when journaling the database.
- 9969
  If a CL client is making changes through an active query as another client is deleting records returned by that query, the CL client can crash.
- 10237
  An ODBC client that mixes calls to SQLBulkOperations and/or SQLSetPos with calls to SQLExecDirect and/or SQLExecute in the same manually committed transaction can crash.
- 10262
  Using the fault tolerant historian sub-system, if the ONLINE field of a LOGARCHIVE record is set from true to null, the archive will be brought online a second time. This causes a problem when a standby RTRDB starts-up and synchronises the archive with the master.
- 10263
  The ODBC driver setup program will report success even if the DLL cannot be copied, due to being in use.
- 10266
  If CL executes the QUIT statement in an inherited create handler, the object will be deleted as if it is an instance of that super class rather than its actual class.
- 10271
  The ODBC SQLStatistics function does not return the correct values for the NON_UNIQUE and ORDINAL_POSITION columns.
- 10398
  It is possible to set a not null column to null if updating the same value twice through an active query using the Callback API.
- 10415
  The LINX transport does not handle failures to send messages due to, for instance, socket buffers being full.
- 10567
  The JDBC driver does not prevent the use of a statement after it has been closed. The effect of using a JDBC statement after it has been closed is undefined.
- 10570
  Setting the ODBC SQL_ATTR_PACKET_SIZE connection attribute before the connection is established, which is when it is allowed, crashes client.
- 10838
  When using the Historian module, if the LOGARCHIVE table is persistent, then archives are not brought back on-line when the RTRDB is restarted.
- 10990
  In CL exporting an attribute using a separate export statement from the definition of the attribute does not work. This can be easily worked around by exporting and defining the attribute with the same statement.
- 11185
  Using Flash DBMS it is not possible to insert rows through an active query that contains an ORDER BY clause.
- 11256
  In Flash DBMS the JOURNALCONTROL table is incorrectly reported as being a system table in the catalogue.

- 11309
  On Windows using the TCP transport a fault tolerant client can occasionally misinterpret a controlled shutdown as a loss of connection and attempt to reconnect. It is therefore important to configure fault tolerant clients with a limited number of reconnection attempts.
- 11325
  A fault tolerant client committing a transaction after a fail-over has occurred has no indication that any locks obtained before the fail-over are no longer held.
- 11338
  In Flash DBMS indexes on local columns do not work correctly.
- 11409
  CL can give the wrong line number for a run-time error on a line containing a function call.
- 11518
  On Windows the CL TcpServer class incorrectly allows a port to be opened that is already in use.
- 11697
  The use of <table>.* in the expression list of an SQL SELECT statement is incorrectly expanded to include columns in other tables listed in the FROM clause.
- 11781
  A client using the LINX or OSE transports can crash if it tries to connect to a journal service rather than the data service of an RTRDB.
- 11800
  It is possible to alter the persistence or local-ness of the DATAPORT table so that it does not match the DATACONNECTION table. This should be rejected.
- 11912
  The Embedded Interface function poly_program_run will crash on POSIX if passed a zero length string for the program name.

7. Polyhedra Support

7.1 Direct Support via the Polyhedra Helpdesk

Polyhedra customers who are not using any other Enea product can obtain support directly from the Polyhedra Helpdesk, by emailing support@polyhedra.com. You can contact us to report a bug, ask for a new feature, or just ask a question.

Customers with more than one Enea product are advised to contact instead the central Enea support desk as described in section 7.2 (below), to ensure the call is handled properly if the underlying issue relates to some interaction between Enea products.

7.2 Support through EneaIssues

Technical support for all Enea licensed and supported products can be requested via EneaIssues (https://eneaissues.enea.com), via e-mail (Worldwide Support - wwsupport@enea.com, North America Support - support@enea.com) or telephone. Addresses and phone numbers for local support can be found at http://www.enea.com/productsupport. You can use EneaIssues to report a bug, ask for a new feature, or just ask a question. An external user's guide for EneaIssues is available at http://www.enea.com/issuetrackingguide.

7.3 Request for Product Support

Before reporting a problem or defect to the Polyhedra Helpdesk or Enea Global Support, please perform the following checks:

- Check the user documentation for the Enea product(s), including trouble shooting sections.
- Check the Polyhedra developer site, http://developer.polyhedra.com, to see if your question is addressed there.
- Check if the issue you request support for is supported by Enea (see the release documentation about which components are included in your delivery).
- Check any information found via http://www.enea.com/productsupport.
- Check whether the problem is specifically related to your application or if it is an error generated by an Enea product.

In order to effectively resolve reported problems, we kindly ask customers to provide us with sufficient information to identify and isolate the specific problem or defect. If you are using another Enea product such as OSE in conjunction with Polyhedra, the release notes for that product describes the information you should supply when submitting a problem report via EneaIssues. If the only Enea product you are using is Polyhedra, the information you need to supply includes

- Problem title (describing the problem, but please avoid using customer specific words).
- Problem description (should be accurate, detailed and explain the problem in terms of Enea product concepts and components. Customer specific abbreviations or words should be avoided or explained).
- Basic information (should be accurate and detailed - mandatory if issue is critical).
    a. How is the problem affecting your business?
    b. Problem impact? On which level in the system does it occur [in operation/upstart/upgrade]?
    c. Is the problem preventing you from shipping your product?
    d. Is the problem located in development, at applications or at an end customer?
- Type of issue: question, probable bug, or new feature request.
- Name and version numbers of the Polyhedra product components (including platform information).
- Error Messages.
- Documented sequence of events to reproduce the problem.

Please also communicate the severity level of this problem or defect for priority purposes. Currently, the following levels of severity are defined:

- **Minor** - The Product(s) functionality operates abnormally. If the Error occurs during the development phase of a Customer's product, the development, integration, or testing is inconvenienced. Alternately, the Customer requires information or assistance with respect to the Product(s) capabilities, installation, or configuration.
- **Serious** - The Product(s) functionality operates with severely reduced capacity causing significant impact to business operations. If the Error occurs during the development phase of a Customer's product, the Error has serious impacts on the development, integration, or testing. A workaround may be available.
- **Critical** (showstopper) - The Product(s) functionality is inoperable causing critical impact to business operations, if the functionality is not restored quickly. If the Error occurs during the development phase of a Customer's product, the Error hinders all of the Customer's development, integration, or testing. No viable workaround is known.

Each reported issue is assigned a problem identification number and is managed using a defect tracking system. Once a problem has been received, you will send a receipt for the issue via email. If using the Polyhedra Helpdesk directly, please reply to the email to give further information, etc; if using EneaIssues, it is best to log on to that system to track the call status and provide further information.

7.4 Software Updates

Major product releases are currently scheduled regularly. Major releases include cumulative upgrades containing corrections to Polyhedra licensed products and new functionality included in the licensed products. Maintenance updates and patches are provided for corrections to known problems and are available upon request.

7.5 Extended Support Services

Extended maintenance options are available for an additional fee that is determined based on the selected services. Potential Extended Support Services could include on-site support, support reviews, and a higher grade of support.