Polyhedra®

Polyhedra 8.9 Release Information

Enea Polyhedra Ltd

ENEA

Copyright notice

Disclaimer

Trademarks

Manual Details

Manual title	**Polyhedra 8.9 Release Information**
Document number	**src/release/readme**
Revision number	**8.9.6**
Revision date	**27th October 2014**

Software Version

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

Version	**8.9.1**

Preface

This document contains release information for Polyhedra version 8.9.

CONTENTS

1. INTRODUCTION 4

2. INSTALLATION 5

3. CHANGES 6

3.1 NEW PLATFORMS SUPPORTED IN 8.9 6
3.2 UPDATED PLATFORMS SUPPORTED IN 8.9 6
3.3 NEW SOFTWARE COMPONENTS IN 8.9 6
3.4 NEW FEATURES AND IMPROVEMENTS IN 8.9 6
3.4.1 SQL Outer Joins 6
3.4.2 SQL GROUP BY and HAVING Clauses 6
3.4.3 SQL DISTINCT Clause 7
3.4.4 SQL IF NOT EXISTS and IF EXISTS Clauses 7
3.4.5 Historian Online Backup 7
3.4.6 Obfuscated Communications 7
3.4.7 Obfuscated CL 7
3.4.8 Login Name 7
3.4.9 Enhanced User Authentication 8
3.4.10 DATACONNECTION and JOURNALCONNECTION Tables 8
3.4.11 Embedded Interface 8
3.4.12 ODBC and JDBC Catalogue Functions 8
3.5 NEW FEATURES AND IMPROVEMENTS IN 8.9.1 8
3.5.1 Login Failure Detection 8

4. MIGRATING TO POLYHEDRA 8.9 9

4.1 SQL RESERVED WORDS 9
4.2 HISTORIAN 9
4.3 LOGIN NAME 9
4.4 USER AUTHENTICATION 9
4.5 OBFUSCATED COMMUNICATIONS 9
4.6 EMBEDDED INTERFACE 10
4.7 JDBC DRIVER 10
4.8 BUILDING POLYHEDRA 10
4.9 BISON 10

5. CORRECTIONS 11

5.1 FULL RELEASE: 8.9.0 11
5.2 1ST MAINTENANCE RELEASE 8.9.1 13

6. KNOWN PROBLEMS 14

7. POLYHEDRA SUPPORT 17

7.1 DIRECT SUPPORT VIA THE POLYHEDRA HELPDESK 17
7.2 SUPPORT THROUGH ENEAISSUES 17
7.3 REQUEST FOR PRODUCT SUPPORT 17
7.4 SOFTWARE UPDATES 18
7.5 EXTENDED SUPPORT SERVICES 18

1. Introduction

This document contains release information for Polyhedra 8.9 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 8.9 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 8.9. The previous release was Polyhedra 8.8.

3.1 New Platforms Supported in 8.9

The following new release kit is included in this release:

- Polyhedra for Enea Linux on ARM (ARMv7) using GNU C/C++

3.2 Updated Platforms Supported in 8.9

The following release kits now support Windows 8.1 and Windows Server 2012 R2 operating systems and Visual C++ 2013 compiler for building Polyhedra client applications and standard components:

- Polyhedra for Win32 on i386 using Visual C++
- Polyhedra64 for Win64 on x86_64 using Visual C++

Visual C++ 2012 continues to be supported, and should be used when building from source.

The supported operating system for the following release kit is now Enea Linux 4.0:

- Polyhedra for Enea Linux on PowerPC P2020 using GNU C/C++

The supported operating system for the following release kits is now OSE 5.7.2:

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

3.3 New Software Components in 8.9

No new software components have been introduced in this release.

3.4 New Features and Improvements in 8.9

The following new features have been introduced in this release:

3.4.1 SQL Outer Joins

Polyhedra SQL now implements the standard explicit join notation that allows joins to be specified in the `FROM` clause of an SQL `SELECT` statement. Using this new syntax it is possible to specify left outer joins, inner joins and cross joins.

Note that right outer joins, full outer joins, natural joins and union joins have not been implemented.

This new feature is described in the Polyhedra *SQL Reference* manual.

3.4.2 SQL GROUP BY and HAVING Clauses

Polyhedra SQL now implements the `GROUP BY` and `HAVING` clauses of the SQL `SELECT` statement. These are used in conjunction with the existing aggregate functions to group the result-set of a query by one or more columns.

This new feature is described in the Polyhedra *SQL Reference* manual.

3.4.3 SQL DISTINCT Clause

Polyhedra SQL now fully implements the `DISTINCT` clause of the SQL `SELECT` statement. Previously it was only implemented for queries that either returned all the primary key columns from the tables they queried, i.e. queries that were guaranteed to return unique rows, or used one or more of the set operators `EXCEPT`, `INTERSECT` and `UNION`.

This new feature is described in the Polyhedra *SQL Reference* manual.

3.4.4 SQL IF NOT EXISTS and IF EXISTS Clauses

Polyhedra SQL now implements an optional `IF NOT EXISTS` clause of the SQL `CREATE` statement and an optional `IF EXISTS` clause of the SQL `DROP` statement that allow existence failures to be ignored.

This new feature is described in the Polyhedra *SQL Reference* manual.

3.4.5 Historian Online Backup

This Historian module has been enhanced to allow a consistent backup of all historical log files and a database snapshot to be generated. This functionality is provided by the new `LOGBACKUP` Historian configuration table

This new feature is described in the Polyhedra *Historic Data Logging* manual.

3.4.6 Obfuscated Communications

All client-server, master-standby and replica communications can now be optionally obfuscated on a per-service basis. The lightweight obfuscation is applied to all messages transferred making it more difficult to determine the structure of the database by analysis of network traffic. Obfuscation is enabled by including the new `obfuscate` service option in the data-service and journal-service names.

This new feature is described in the *Polyhedra User's Guide* manual.

Note that this does not provide the same level of protection as that provided by the Polyhedra SSL transport.

3.4.7 Obfuscated CL

A new form of compiled binary CL file is now supported that allows CL for a Polyhedra application to be supplied in an obfuscated form that is not readily open to inspection or modification by the users of the application.

Both the CL and RTRDB components can now generate binary CL files that correspond to individual CL source files and load a combination of multiple source and binary CL files. The new *cl_development* resource, supported by both the CLC and RTRDB components, enables the generation of binary CL files for all CL source files as they are loaded. The existing *cl_library* resource has been enhanced so that specific source or binary CL files can be loaded or whichever is newer of a corresponding pair of source and binary files.

This new feature is described in the *CL Reference* manual.

3.4.8 Login Name

A new optional `LOGIN_NAME` column has been added to the existing `USERS` table. This provides a login name for the user that may be different from the name of the user stored in the `NAME` column and that may be changed in a running system.

Note that the existing `NAME` column of the `USERS` table will continue to be used for the following:

- The `default_user` resource
- The `OWNER` column of the `TABLES` catalogue table
- The `USERNAME` column of the `DATACONNECTION` table
- The `GRANT` and `REVOKE` SQL statements
- The `GetUser` SQL function
- The `GetUser` CL function
- The `user_name` argument to the `poly_log_operation` Embedded API function

This feature is described in the *Real-Time Relation Database* manual.

3.4.9 Enhanced User Authentication

User authentication has been enhanced with support for a challenge-response authentication mechanism designed to protect against replay attacks. This is enabled by setting the existing `password_security_level` RTRDB resource to a value of 2. This also specifies that user passwords are stored as salted hashes in the `PASSWORD` column of the `USERS` table.

The existing `SQLEncryptPassword` ODBC API function, `ClientAPI::EncryptPassword` and `ClientAPI_EncryptPassword` Callback API functions and `encryptPassword` CL function can now generate encrypted passwords (salted hashes) suitable for storing in the `PASSWORD` column of the `USERS` table when the password security level is set to 2.

This feature is described in the *Real-Time Relation Database* manual.

3.4.10 DATACONNECTION and JOURNALCONNECTION Tables

A new `OBFUSCATED` column has been added to the existing `DATACONNECTION` and `JOURNALCONNECTION` tables. The new columns indicate whether a client connection is obfuscated.

This is described in the Polyhedra *Utility Classes* manual.

3.4.11 Embedded Interface

The following new user-supplied function has been added to the Polyhedra Embedded Interface:

```
int poly_random_data(
        unsigned char *     buffer,
        int                 length);
```

It is called whenever a Polyhedra component requires random data as part of user authentication, i.e. to generate the salt and challenge.

This is described in the *Polyhedra on Embedded Systems* manual.

3.4.12 ODBC and JDBC Catalogue Functions

The performance of the ODBC and JDBC catalogue functions has been significantly improved. Although not required, the following indexes can be added to achieve further performance improvements, but note that this advice may change in future versions of Polyhedra.

```
CREATE ORDERED INDEX cat_index_1 ON attributes ( table_name );
CREATE INDEX cat_index_2 ON indexes ( table_name );
CREATE ORDERED INDEX cat_index_3 ON indexattrs ( index_name );
```

3.5 New Features and Improvements in 8.9.1

The following new feature has been introduced in this release:

3.5.1 Login Failure Detection

A new `LOGIN_FAILURE_COUNT` column has been added to the existing `DATACONNECTION` table. The new column is incremented by 1 whenever a login attempt fails.

This is described in the Polyhedra *Utility Classes* manual.

4. Migrating to Polyhedra 8.9

This section details how to migrate an application from an earlier version of Polyhedra to Polyhedra 8.9.

Except as described below, Polyhedra 8.9 does not introduce any incompatibility with and is interoperable with Polyhedra 8.8. Polyhedra 8.9 will act as a maintenance release for 8.8. There will be no further 8.8.X maintenance releases.

4.1 SQL Reserved Words

Polyhedra 8.9 introduces ten new SQL reserved words `CROSS`, `FULL`, `IF`, `INNER`, `JOIN`, `LEFT`, `NATURAL`, `OUTER`, `RIGHT` and `USING`. Existing applications that use these words as identifiers should either be modified to rename them or use delimited identifiers to access them. If this is not possible, the `sql_disable_v8_9_reserved_words` resource can be used to disable the new reserved words, and the functionality associated with them. Note that this resource is provided as a migration aid and may be withdrawn in future versions of Polyhedra.

We advise that all new applications, or changes made to existing applications, avoid the use of SQL:2003 reserved and non-reserved words as identifiers. Use of delimited identifiers can also reduce the likelihood of conflicts with any reserved words added to future versions of Polyhedra.

There is a tool on the Polyhedra developer site (http://developer.polyhedra.com) to help check whether existing databases use names that could cause problems, including SQL reserved words introduced in Polyhedra 8.9 and words that are of special meaning in SQL:2003.

4.2 Historian

A new historian configuration table `LOGBACKUP` has been added in Polyhedra 8.9. A Polyhedra 8.9 RTRDB supports loading a database load file containing Historian configuration with or without the `LOGBACKUP` table defined. If the `LOGBACKUP` table is not present in the database load file, a Polyhedra 8.9 RTRDB will behave identically to a Polyhedra 8.8 RTRDB.

Migration of an existing database load file containing Historian configuration to use the new `LOGBACKUP` table requires it to be added using the SQL `CREATE TABLE` statement and the RTRDB to be restarted or, if in fault tolerant configuration, failed-over.

4.3 Login Name

Migrating an existing application to use the new `LOGIN_NAME` column of the `USERS` table requires the following steps:

1. Altering the `USERS` table to add the new `LOGIN_NAME` column, but without a `NOT NULL` constraint.
2. Populating the `LOGIN_NAME` column with values copied from the `NAME` column.
3. Altering the `LOGIN_NAME` column to add a `NOT NULL` constraint.
4. Restarting the RTRDB or, if in fault tolerant configuration, failing-over.

If the application is only allowing the name of the special SYSTEM and PUBLIC users to be changed, it may be appropriate to have CL code to automatically set `LOGIN_NAME` to `NAME` when a record is inserted into the `USERS` table.

4.4 User Authentication

If a pre-Polyhedra 8.9 client attempts to authenticate a user when connected to a Polyhedra 8.9 RTRDB that is using password security level 2, the authentication will be failed.

When changing the password security level used by the RTRDB it is necessary to reset all passwords stored in the `PASSWORD` column of the `USERS` table to be consistent with the new level.

4.5 Obfuscated Communications

If a pre-Polyhedra 8.9 client attempts to connect to a Polyhedra 8.9 RTRDB using a service that specifies the `obfuscated` option, the connection will be rejected.

4.6 Embedded Interface

Existing applications that use the Polyhedra Embedded Interface will need to provide implementations of the new `poly_random_data` function.

4.7 JDBC Driver

The Polyhedra JDBC driver package is now a .jar file rather than a .zip file.

4.8 Building Polyhedra

Building the CLC component now requires the inclusion of a new polyzlib library in the link stage. The same applies to the RTRDB component when including either the CL or Historian modules. The build instruction included in the release kits give precise instructions for linking these components.

4.9 Bison

Building Polyhedra 8.9 from source now uses the GNU parser generator bison on all platforms. Previously bison was only used for building Polyhedra for 64-bit Windows and yacc was used for all other platforms.

Note that this usage of bison has no implications with regard to FOSS licences. Please see the *Open Source Repor*t included in the release kit for details.

5. Corrections

This section details the corrections that have been made to Polyhedra 8.9.

5.1 Full Release: 8.9.0

All bugs fixed in Polyhedra 8.8 up to the most recent maintenance release 8.8.2 have been checked against Polyhedra 8.9 Release and, where appropriate, are also fixed in this release. The following bugs present in 8.8.0 and earlier are fixed by this release:

- 4707
  An active query would not notice changes of order if the ORDER BY columns are not mentioned elsewhere in the query.
- 6198
  The ODBC connection attribute SQL_ATTR_POLY_FT_RECONNECTION_TIMEOUT was not implemented.
- 8419
  Historical compressed log files did not handle null values correctly.
- 9481
  The historian resource log_long_timestamp was not applied to compressed log files.
- 10275
  If an archive on a compressed stream was produced on the master side of a Fault Tolerant pair as a standby or replica was starting up then the archive on the standby or replica side could have less data in it than the master side.
- 10390
  An active query would not generate a delta when a field changed if that field was only referenced in a non-trivial output expression.
- 10673
  When using the historian module of the RTRDB on OSE the log files were not created with the prefix 'log-' as on all other platforms.
- 11190
  The ODBC catalogue inspection functions could be slow when the database schema was large.
- 11436
  On OSE, when starting a load module via a Polyhedra component, a long path to the load module (over PM_INSTALL_HANDLE_LENGTH) could result in a non-unique install handle, which on second attempt could cause the component to hang.
- 11554
  The RTRDB could crash if a table was created containing a column that was an array of a table having a name starting with "binary".
- 11556
  The RTRDB could crash when a record was inserted into a table that derived from a table containing a domain with default values.
- 11558
  Creating a view that uses a transient not null column without a default from any of its base tables would generate an RTRDB warning of the form: "Warning: Not null transient column <column> found in persistent table <view>, please add a non-null default.". This bug was introduced in Polyhedra 8.8.
- 11563
  Incorrect warnings were generated about not null transient domain columns.
- 11580
  The CL activate handler was not being called on the JOURNALCONTROL table when the RTRDB started-up in standalone or master mode. This bug was introduced in Polyhedra 8.8.

- 11581
  It was not possible to grant privileges on the JOURNALCONTROL table. This bug was introduced in Polyhedra 8.8.
- 11582
  If an object query from a Call-back API client failed due to lack of privileges, the debugger 'queries' command could crash the RTRDB if the query explicitly defined the output columns. This issue was long standing and did not apply to CL object queries.
- 11590
  It was not possible to login from the debugger if password_security_level was set to 1.
- 11594
  On Windows, CL type coercion from Real to String could give incorrect results for some values.
- 11604
  The per-platform linking instructions for the RTRDB have been improved. Some missing libraries have been added and the libraries required to build a minimal functionality RTRDB have been corrected.
- 11606
  Added documentation of tcp_message_cache_size resource in RTRDB manual.
- 11607
  It is now possible to for historical archive files from one stream to be brought online on another stream as long as the column structure is compatible. Previously this was only allowed if the streams were on same source table.
- 11609
  The 64-bit versions of the Callback and ODBC libraries did not correctly convert the database (32-bit signed) integer data type to string. If the value was negative it was converted to a large positive value.
- 11610
  In a fault tolerant configuration using the historian component the average and integral compression fields used in a historical compressed stream could be incorrect for the compression slot being filled when a failover occurred.
- 11612
  Active queries on historical data did not always bring back all the records when compared with an identical static query.
- 11616
  A referential action of CASCADE DELETE would fail to execute correctly if the column it was defined on also had a NOT NULL constraint.
- 11636
  A referential action of CASCADE or SET NULL could be performed not only on the correct records but also on other records in the table that referenced different targets.
- 11638
  CL would crash when a CL query returned more than one column whose type did not match the type of the attribute being filled.
- 11643
  The RTRDB would crash, if using the Historian component, when disabling logging on a stream if the table being logged contained records where the column being used as the name column contained a null value.
- 11646
  The historian component of the RTRDB would crash if an object being logged was inserted with its name column containing " which was subsequently updated in a separate transaction to have a non empty name, 'name' say. If an object was inserted with a valid name in the name column and the value was then updated to " or null and then deleted in the same transaction the historian did not log the delete. If an object was inserted with a valid name in the name column and then that value was updated to another valid name and then deleted in the same transaction then the historian incorrectly logged two deletes for the first name and none for the second.
- 11651
  SQL LIKE did not accept an expression as the pattern.
- 11652
  In a fault tolerant configuration the standby RTRDB would crash if an integer64 column was updated via an SQL call statement using a numeric literal parameter value < 2^32.

- 11656
  The combination of an aggregate function and a non-aggregate column reference in the same expression could crash the RTRDB.
- 11660
  The RTRDB could crash when loading binary CL containing a script for a domain.
- 11667
  Updates to domain columns were not journalled to a standby or replica if the transaction also included updates to other non-domain columns.
- 11696
  On very rare occasions a replica RTRDB with a very small heartbeat interval (e.g. journal_heartbeat_interval=250) could crash on start-up.
- 11707
  On rare occasions the RTRDB could crash when trying to recover from memory exhaustion during a database save.
- 11712
  An SQL condition of the form 2 < a (constant expression first) that used an ordered index could return the incorrect rows.
- 11716
  On rare occasions the RTRDB could hang if a SAVE or SAVE INTO statement was executed from an ODBC client when other transaction were being executed.
- 11741
  The RTRDB would crash if the buffer size of an historical log file was set too small. This would occur when resizing the log file by disabling it, setting the BUFFERSIZE column of the LOGDATA table with too small a value and then re-enabling it.
- 11747
  An SQL query with a WHERE clause containing a restriction of the form "a < 1 and a > 1" incorrectly returned a row when column ‘a’ had an ordered index.
- 11759
  Altering a table, which has a view defined on it, to be not local on a master RTRDB would cause any connected standby or replica RTRDB to crash.

5.2 1st Maintenance Release 8.9.1

- 11765
  The default data service connection used by CL in the RTRDB did not have SYSTEM access when security was enabled.
- 11768
  The merge utility could fail to merge historian archives of a compressed stream if a raw LOGCOLUMN of the stream had INDEXMETHOD set.
- 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 to return all the correct records.
- 11806
  The RTRDB would incorrectly load a corrupt database load file that contained duplicate records.

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.
- 6187
  The Callback API function TransAPI::DeleteTrans cannot be called if the connection has been lost. This can result in a memory leak.
- 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.
- 11485
  Polyhedra SQL incorrectly rejects identifiers containing consecutive underscores.
- 11518
  On Windows the CL TcpServer class incorrectly allows a port to be opened that is already in use.
- 11546
  It is possible to alter the DATACONNECTION table to be local when the DATAPORT table is non-local. This should not be allowed.
- 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.

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.