# Interbase Replication Suite Guide

**Editor:**
Tomáš Mandys, tomas.mandys@2p.cz (2p plus)

**Home site:**
http://www.2p.cz

# Introduction

Plenty of software uses a database to store data. There are many types of database implementations but generally two main branches exist - SQL based relation database management systems (RDBMS) and file based database.
One of the RDBMS is an Interbase. The Interbase is powerful transaction SQL database engine that implements many advanced features - views, triggers, stored procedures, generators, etc. Interbase was developed by Borland as commercial database but version 6.0 was released as open-source. It means that it's possible use it for-free for any purposes. On open-source database was founded new branch named Firebird. Even Borland released next Interbase v7 as non open-source, the Firebird clone remains open-source.
Even the Interbase architecture is robust and powerful, comparable to other concurrent, one feature limits it - there is not implemented replication engine such as in Oracle, MSSQL or even MSAccess.
In many cases data of an enterprise are not stored in one database but they are distributed to more locations. Replication engine try to maintain data in each of databases as synchronous as possible. To be replication efficient, the engine catches only data changes and passes them to the other databases.
There are many reasons why to split data:

- non-reliable connection to main database
- security, shadow database in DM zone
- data backup to remote *shadow* database at different place
- external database on laptops for representatives
- etc.

There are many replication models because two enterprises will have probably different requirements.
Lets see several examples.
There are main database and shadow (read-only) database that serves as backup. When the main database crashes, *shadow* database is activated as main one.
There are more equivalent databases. Data should be the same in each of them.
Enterprise maintains net of agents that have database at personal laptops. They can change records (make orders, update information). And because they are competitors they can see only own data.
Enterprise must consolidate data daily from subsidiaries
So there are many questions:

- how often replicate?
- does exist reliable and fast online connection?
- all tables?
- all records in table?
- uni or bidirectional?

Because replicator must meet all contradictory requirements, changes fire triggers that log them in source database and then they are replicated to targets. It means that are transferred only changes. Due to referential integrity it's necessary replicate in the same sequential order as the changes occurred. If new records may be inserted in more locations, primary keys must be unique - every record must be identifiable by unique primary key. The logging brings overhead and the source database growths, so replication log records are deleted when are successfully replicated. You can configure what fields values log - generally primary key values must be logged mandatory, *NOT NULL* and foreign key fields recommended and remaining values optionally.

Let's suppose that the replicator connects to the source database, looks to change log and if there are some records connects to the target database and transfers the changes. One replicator is connected to both databases in one time, hence it's **online replication**. But what happen if connection is broken before transfer is finished? Nothing dangerous, change log record is removed when change is committed in target database - it simulates two phase commit. But when an constraint problem occurs record is moved to other log where are stored conflicting records. The replicator cannot solve such problem automatically, it must be solved manually, hence the log is named **manual log**.
How is possible replicate when one of databases is unavailable from opposite location? It is hidden behind the firewall, machine where is running connects via dial-up or connection is unreliable. The solution is **offline replication**. In this case the first replicator is connected to source database and prepares packages containing replication data and passes them to a shared drop-off location, the second replicator reads data from reads the shared location and applies data to the target database. The offline replication uses robust four-phase acknowledge protocol and stores package transfer history if **transfer log**.
In ideal world all data are correctly replicated. But in real world from time to time happen an exception or an accident (hard disk crashes, replication integrity corrupts, etc.). Even no crash happen there is one exception - start of replication, because you must setup data to be synchronous before replication is done. Who can manage databases to be ready for replication? Answer is a **synchronization**. The replication transfers only changes whereas the synchronization compares databases, table to table, record to record. This process is of course much more time consuming especially for large databases. Note that synchronization is always online.
Now let's suppose that there is an table containing very important data and somebody intentionally rewrites it. This problem solves **history** - replicator keep record history in **history log**. Unlike the replication the history is not replicated to other database.
Another possibility how to keep record history is **incremental SQL log**. It's a plain text file containing sequential list of SQL commands. Note even Interbase data manipulation language does not support BLOB values, the incremental log contains also BLOB metadata. Since it is plain SQL text file it can be used to feed changes to any SQL database, e.g. you can replicate changes to MySQL. Both the online and the offline methods are supported.
If data are changed at both sides data inconsistency can occur, when corresponding record is changed in the source database and the target database. A **field/record conflict** happened. The replicator is able to recover conflict automatically according or to pass conflict to the manual log. The recovery is configurable for every relation.
All the configuration data, i.e. schemes are stored to a separate **configuration database**. You can prepare it once and use for all locations, does not matter if you access it online or make physical copy. It simplifies administration significantly.

## Features

- the Interbase and Firebird (1.0, 1.5) databases supported
- the SQL Dialect 1 and 3 supported
- the offline or online synchronization
- FTP, email or shared net dir transfer for offline replication, open source transfer libraries
- the offline packages can be encoded for improved security or compressed, open source interface, CRC checking
- visual replication manager (scheme editor)
- custom configurable history of records
- the replication server running as a GUI application or a console/NT service application

- multi-threading supported
- multi-segment primary key supported
- conditional replication and synchronization
- extended conflict checking
- replication into tables or stored procedures
- replication actions and errors are logged into logs
- replication statistics
- international support - optional UTC time stamps, Unicode/UTF-8 encoding
- source/target database recovery from daily backup and incremental backup
- SDK - a developer can implement full replication server functionality to own software
- password encryption
- database backup/restore
- database cloning

## Features not yet implemented

- replication of array fields (not planned)
- not tested for Asian code pages
- replication of structure (DDL)
- record history viewer
- GUI driven database comparer (dbdiff)
- custom installer NSIS support

## Requirements

- installed Interbase/Firebird client
- the SQL server must support *CommitRetaining* and *RollbackRetaining* (Interbase v6)
- Windows 9x/ME/NT/2000/XP or Linux at x86 platform

# History

Here is brief history of development progress.

### v.2.0 (6/2004)

- required IB_Repl UDF library
- Professional version (commercial)
- rewritten GUI
- NSIS installator
- replication tasks defined in replication server
- server running as NT service - IBReplServerSvc
- IBReplInst
- IBReplPackage2XML
- rewritten SDK, VCL components
- replication environment
- UTF-8 offline package support
- conflict logging
- password encoding
- environment and macro expansion
- *disable replication triggers* feature
- replication to target REPL$LOG and from target REPL$LOG
- record history
- extended conflict checking
- custom fields
- log file rotation
- conditional synchronization
- multi-threading support
- database cloning
- backup/restore
- stored procedure skeleton generator
- automatic (config) database upgrade
- replication cron
- external file lock
- etc.

### v.1.6.9.1 (4/2004)

- added *Add/remove server user* commands to IBReplMan
- fixed bug when replicating DATE field in primary/foreign key

### v.1.6.8.1 (11/2003)

- fixed offline transfer bug - numeric fixed point numbers was exported in package with national (non-dot) decimal delimiter
- new LogErrSQLParams property
- improved reconnection when DB connection was lost
- package CRC checking
- auto offline package resending

- improved logging
- improved efficiency for larger schemes
- added Default character set to IBReplServer

### v.1.6.7.1 (10/2003)

- fixed possible SNAPSHOT table dead-lock when WIPELOGONDELETE used

### v.1.6.6.1 (9/2003)

- fixed remapping offline package fields problem
- do not read views into relation list

### v.1.6.5.4 (9/2003)

- fixed possible dead-locking due `UPDATE REPL$SNAPSHOT` in trigger `TG_I_REPL$LOG_SEQID`

### v.1.6.5.3

- binary BLOBs support in SQL log
- new protected methods in TIBReplDataModule to encode/decode BLOB SQL

### v.1.6.5.1 (5/2003)

- ibreplscr command line tool
- REPL$SNAPSHOT table in source and target database
- new function TIBReplDataModule.ExecSQLScript
- see upgrade.txt

### v.1.6.4.3

- FDB extension (default extension of Firebird 1.5)
- optional ibrepl.key registration
- ibreplc ported to Linux

### v.1.6.1 (5/2003)

- online replication bug fixed (when SQL dialect is 3) (SELECT * FROM LOG2)
- obsolete DBID field bug in IBREPLMAN manual log
- ReadTimeout parameter in transfer_email and ftp library

### v.1.6 (3/2003)

- offline replication
- SQLDialect 1
- software development libraries
- visual replication manager
- command line replicator

### <= v.1.5 (2002)

- synchronization and online replication
- Firebird/Interbase

# Registration

Registration is required for each computer where replication server (p.28) or console replication server (p.38) is running. Registration is also required for SDK (p.52). Please follow instructions in the registration dialog.
You can register using online registration form (http://www.2p.cz/en/ibrepl/registrace.html). You'll receive activation key immediately to your mailbox.

## Interbase Replication Manager

The GUI tool that administers replication model. The model consists of schemes. The schemes are stored in a configuration database. It is common database of Interbase format therefore configuration may be maintained also using plain SQL commands.

**See also:**
Log (p.8)
Schema editor (p.8)
Statistics (p.14)
Settings (p.15)
Command line (p.8)

## Replication manager - Command line parameters

A command line options can override default behavior. The options have higher priority than an INI file options.

**/I:config_filename**
Overrides the default INI file (`IBREPL.INI` in a program directory).

**/E:<name>=<value>**
Defines a key for an environment (p.37)

**/TRACESQL**
Enables tracing/profiling of SQL commands using a SQL monitor

## Log tab

**Settings** (p.8) **Schema editor** (p.8) **Statistics** (p.14)

The manager is logging messages to a terminal window and transparently to a file, see Settings/Log (p.16).

### Menu commands

**Clear log**
Clear the log terminal window

## Schema editor tab

**Settings** (p.8) **Schema editor** (p.8) **Statistics** (p.14)

Databases (p.10) Schema (p.11) Groups (p.11) Databases in group (p.12) Relations (p.13) Fields (p.14)

The schema editor is main windows where configuration tasks are performed. When tab is focused connect to a configuration is performed. So first define the configuration database (p.15). The tab contains five database grids connected to the configuration database. All the configuration is performed on database in separate transaction. So commit changes to take effect.
The is side effect - if a replication task is running then statistics fields of *SCHEMATA* and *RELATIONS* tables are updated periodically - record lock possible. Stopping of replication when configuring is recommended.

## Configuration database menu commands

**Commit**
Commit all changes to the configuration database

**Rollback**
Rollback all changes in the configuration database

**Open database**
Open the configuration database

**Close database**
Close the configuration database

**Create database**
Create new configuration database according to settings (p.15).

**Upgrade database**
Upgrade the configuration database that was created by a former version of Interbase Replicator. The upgrade process is safe, no data should be lost. Note that upgraded database won't be compatible to older Interbase Replicator.

**Backup**
Backups configuration database to `GBAK` compatible format.

**Restore**
Restores configuration database from `GBAK` compatible format. If database already exists data in the database are replaced and data lost.

**Environment**
Edit `ENVIRONMENT` table of the configuration database, see environment (p.37)

**Settings**
Open global settings dialog (p.15)

# Schema editor / Databases

**Settings** (p.8) **Schema editor** (p.8) **Statistics** (p.14)

Databases (p.10) Schema (p.11) Groups (p.11) Databases in group (p.12) Relations (p.13) Fields (p.14)

The grid browses records of DATABASES table.

## Database menu commands

**Insert**
Create a new database registration record, see Database properties (p.17)

**Edit**
Edit a database registration record, see Database properties (p.17)

**Delete**
Delete a database registration record

**Environment**
Edit `REPL$ENVIRONMENT` table of given database, see environment (p.37)

**Upgrade database**
Upgrade a database that was used with older version of Interbase Replicator. The upgrade process is safe, no data should be lost. But it's recommended replicate all data before upgrade process is started. Note that upgraded database won't be compatible to older Interbase Replicator.

**Create system objects**
Creates / updates all system (replication) objects of given database. It prepares database for each schema in which the database participates. It also grants rights to dedicated replication user.

**Drop system objects**
Removes system objects of all schemes from database. All replication objects are dropped and replication records are lost Some dependency errors may appear.

**Drop triggers**
Removes only replication triggers of all schemes, helpful when replication condition was changed. No replication records are affected.

**Add server objects**
Creates server objects, SQL users defined in schemes as replication user.

**Drop server objects**
Removes server objects - SQL users defined in all schemes as replication user.

**Backup**
Backups database to GBAK compatible format.

**Restore**
Restores database from GBAK compatible format. If database already exists data in the database are replaced and data lost.

## Schema editor / Schema

**Settings** (p.8) **Schema editor** (p.8) **Statistics** (p.14)

Databases (p.10) Schema (p.11) Groups (p.11) Databases in group (p.12) Relations (p.13) Fields (p.14)

The grid browses records of SCHEMATA table.

### Schema menu commands

**Insert**
Create a new schema record, see Schema properties (p.18)

**Edit**
Edit a schema record, see Schema properties (p.18)

**Delete**
Delete a schema record

**Source database**
Edit source database (p.19) record. Every schema has assigned one source database and one or more target databases (in one or mere groups).

## Schema editor / Groups

**Settings** (p.8) **Schema editor** (p.8) **Statistics** (p.14)

Databases (p.10) Schema (p.11) Groups (p.11) Databases in group (p.12) Relations (p.13) Fields (p.14)

The grid browses records of SCHEMADB table where `DBID=0`.

### Group (S) menu commands

**Insert**
Create a new group of target databases, see Target database properties (p.19)

**Generate PK,FK, data fields**
Source database (of selected schema - SchemaID and GroupID) is opened (as set in `DATABASES`) and its objects are read (tables, fields, primary and foreign keys) and then are written to `RELATIONS` and `FIELDS` tables. Only objects that did not exist in configuration database are added.

**Create system objects**
It generates all replication objects (triggers, tables, user) of given schema in source database. Use also for refreshing of objects - only objects that does not exist in source database are created.

**Drop system objects**
It destroys all replication objects of given schema.

**Drop triggers**
It destroys all replication triggers of given schema in source database. Use when a relation condition was changed (*Drop triggers* + *Create system objects*). No replication records are affected.

**Clean source data**
Delete all replication record that belong to selected schema. `REPL$LOG`, `REPL$MAN`, `REPL$TRANSFER`, `REPL$FIELD` are affected. Note that offline packages must be deleted manually.

**Add server users**
It adds SQL server user for given schema. If the user already exists then only password is updated. Note that Firebird does support connecting to service manager only for *superserver*.

**Remove server users**
It removes SQL server user of given schema.

## Schema editor / Databases in group

**Settings** (p.8) **Schema editor** (p.8) **Statistics** (p.14)

Databases (p.10) Schema (p.11) Groups (p.11) Databases in group (p.12) Relations (p.13) Fields (p.14)

The grid browses records of `SCHEMADB` table where `DBID<>0`.

### Group (T) menu commands

**Insert**
Create a new target database record, , see Target database properties (p.19)

**Edit**
Edit a target database record, , see Target database properties (p.19)

**Delete**
Delete a target database record. If database is last remaining database in a group the group is deleted.

**Create system objects**
It generates all replication objects (tables, rights) in target database. Use also for object refresh - only objects that does not exist in target database are created.

**Drop system objects**
It drops all replication objects from target database.

**Clean source data**
Delete all replication record that belong to selected target database. `REPL$LOG`, `REPL$MAN`, `REPL$TRANSFER`, `REPL$FIELD` are affected. Note that offline packages must be deleted manually.

**Create stored procedures**
Creates skeletons of target database stored procedures, i.e. where target relation is selected as *stored procedure*. The administrator then will implement functionality by himself.

**Add server user**
It adds SQL server user for selected schema. If user already exists only password is updated. Note that Firebird does support connecting to service manager only for *superserver*.

**Remove server user**
It removes SQL server user of given schema.

**Clone from source**
It clones a source database, i.e. removes all replication objects, replaces target database and creates replication objects. If target database already exists is replaced and data are lost.

**Clone empty from source**
It clones a source database, i.e. removes all replication objects, replaces target database and creates replication objects. No data is added to new database. If target database already exists is replaced and data are lost.

## Schema editor - Relations

**Settings** (p.8) **Schema editor** (p.8) **Statistics** (p.14)

Databases (p.10) Schema (p.11) Groups (p.11) Databases in group (p.12) Relations (p.13) Fields (p.14)

The grid browses records of `RELATIONS` table.

### Relation menu commands

**Insert**
Create a new relation record, see Relation properties (p.21)

**Edit**
Edit a relation record, see Relation properties (p.21)

**Delete**
Delete a relation record and all dependent field records

**Copy from schema**
It copies relation and field records from other schema. Only non existing records are added (according to `RELATIONID`), see Copy from dialog (p.17)

# Schema editor / Fields

**Settings** (p.8) **Schema editor** (p.8) **Statistics** (p.14)

Databases (p.10) Schema (p.11) Groups (p.11) Databases in group (p.12) Relations (p.13) Fields (p.14)

The grid browses records of `FIELDS` table.

## Field menu commands

**Insert**
Create a new field record, see Field properties (p.24)

**Edit**
Edit a field record, see Field properties (p.24)

**Delete**
Delete a field record

**Copy from relation**
It copies field records from other relation. Only non existing records are added (according to `FIELDID`), see Copy from (p.17)

# Statistics

**Settings** (p.8) **Schema editor** (p.8) **Statistics** (p.14)

Number of replicated records, conflicts, errors and elapsed time are written to statistics fields of `SCHEMATA` and `RELATIONS` tables.

## Statistics menu commands

**Clear schema statistics**
Clear all statistic fields of selected schema.

**Clear group statistics**
Clear all statistic relation fields of selected group.

**Clear relation statistics**
Clear all statistic fields of selected relation.

## Replication Manager - Settings

**Settings** (p.15): Config.database (p.15) Appearance (p.16) Log (p.16) Replication (p.16)

Manages a global manager configuration that is shared with server configuration (p.29).
In a bottom section is indicated current INI file.
**See also:**
Environment (p.37)

## Settings / Configuration database

**Settings** (p.15): Config.database (p.15) Appearance (p.16) Log (p.16) Replication (p.16)

**Database file name**
A name of the configuration database is entered in Interbase convention:

| | |
|---|---|
| TCP/IP | `server_name:path/filename` |
| NetBEUI | `\\server_name\path\filename` |
| SPX | `servername@path/filename` |

If a multi-threading replication is used with a local Interbase server, use *"localhost:/path/database.gbd"* syntax. It's limitation of an Interbase client. For more information see Jeff Overcash' messages in an IBX newsgroup. Environment macros (p.37) may be used.
For example:

```
SERVER1:C:/DATA/DATABASE.GDB
localhost:$(_DIR_)/DATA/DATABASE.GDB
```

**SQL dialect**
A dialect used for connecting or creating of a config database. If the SQL dialect is `3` then database object names are case sensitive and can contain spaces.

**User name, Password**
A user name and password that the replication server uses to login in the configuration database. The user must have granted enough rights, to read (SELECT) all the tables and modify (UPDATE) statistic fields (*S_xxx*). The password may encoded to be human unreadable using * button. Environment macros (p.37) may be used.

**Object prefix**
A prefix for objects used in a configuration database. Not necessary unless more configuration databases is situated in one *GDB*.

**Def.char set**
A character set of a configuration database. Is important primarily when database is created.

## Settings / Configuration database

**Settings** (p.15): Config.database (p.15) Appearance (p.16) Log (p.16) Replication (p.16)

**Max.lines in terminal**
Max. lines logged to a terminal window. Too many lines decrease performance significantly.

## Settings / Log

**Settings** (p.15): Config.database (p.15) Appearance (p.16) Log (p.16) Replication (p.16)

**DB log**
A log where are logged massages considering to administration tasks

**File name**
A file where messages are logged. Set `NUL` to disable file logging. Environment macros (p.37) may be used.

**Max.log size, Rotate log count**
If size of a log file exceed *Max.log size* then if a *Rotate log count* is zero/empty the file is deleted else the next rotate log is used. Max.number of possible log files determines *Rotate log count*. Set *Max.log size* to `0` to allow unlimited size of the log file.

**Log error SQL commands**
Log full SQL commands when an error is raised. Its helpful to debug SQL DML (data manipulation language) errors.

**Log error SQL params**
Log SQL command parameter values when an error is raised. Its helpful to resolve referential integrity conflicts.

**Verbose log**
Log more messages when SQL DML (data manipulation language) or backup/restore (all output from `GBAK` is logged) is performed.

## Settings / Replication

**Settings** (p.15): Config.database (p.15) Appearance (p.16) Log (p.16) Replication (p.16)

**Now as UTC**
If key is checked then time stamps are logged as UTC (Universal Time Coordinates)/GMT (Greenwich Mean Time).

**Max.key length**
Length of VARCHAR fields where are stored primary keys in `REPL$LOG` and `REPL$MAN` tables. Must be larger than max. possible length of concatenated multi-segment primary keys (each segment separated by separator) in any of replicated tables.

## Copy from dialog

Set object identifier to be copied from.

**See also:**
Schema editor (p.8)

## Database properties

The dialog edits single record of `DATABASES` table.

**DBId**
A unique database identifier greater than zero, if database is already used in a scheme then identifier cannot be changed.

**Type**
- *database* - common Interbase database, typical extension are `.GDB` or `.FDB`
- *text file* - plain text file where are logged SQL command `.TXT` or `.SQL`

**Name**
A descriptive name

**File name**
If type is *database* then it's entered in Interbase convention:

| | |
|---|---|
| TCP/IP | `server_name:path/filename` |
| NetBEUI | `\\server_name\path\filename` |
| SPX | `servername@path/filename` |

If a multi-threading replication is used with a local Interbase server, use *"localhost:/path/database.gbd"* syntax. It's limitation of an Interbase client. For more information see Jeff Overcash' messages in an IBX newsgroup. The password may be encrypted to be human unreadable using * button. Environment macros (p.37) may be used.
For example:

```
SERVER1:C:/DATA/DATABASE_$(DBID).GDB
localhost:$(_DIR_)/DATA/DATABASE_$(DBID).GDB
```

If type is *text file* then it's a file name entered in system networking convention `\\server_name\resource\filename`
For example:

```
\\SERVER1\\DATA\\LOG\\BACKUP_$(DBID).SQL
```

**Admin user, password, role**
Data which is used by replication manager when reading database structure, creating of replication tables and triggers. User must be of course registered in Interbase server registration database (e.g. using *IB Console*).
The most common *Admin user* is `SYSDBA`.

**SQL dialect**
A dialect used for connecting to the database. If the SQL dialect is `3` then database object names are case sensitive and can contain spaces.
If type is *text file* then output SQL commands are compliant to given SQL dialect.

**Char.set**
Implicit database code page. Important for correct string translation.

**Object prefix**
All replication tables in source/target databases will be prefixed with this name. Default value is *REPL$*. Ensure names do not contain spaces if SQL dialect `1` is used.

**Custom fields**
Any number of custom fields may be added to replication tables of database and the customs fields are maintained by manager. Every custom field resides an individual line in following format:

```
table_name|field_name|field_type
```

Lines starting with semicolon (';'), sharp ('#') or slash ('/') are comments. Note that table names are defined without the object prefix.
In example are added `OWNER` fields to `REPL$LOG` and `REPL$HISTORY` tables to log current database user:

```
# log to repl log current SQL user (owner of record)
LOG|OWNER|VARCHAR(30) DEFAULT USER
HISTORY|OWNER|VARCHAR(30) DEFAULT USER
```

**Comment**
Any free comment

## Schema properties

The dialog edits single record of `SCHEMATA` table.

**SchemaId**
A unique scheme identifier greater than zero. If scheme is already used in a group then identifier cannot be changed.

**Type**
replication changes are stored in source database `REPL$LOG` table to be replicated to target databases
record history changes are stored in source database `REPL$HISTORY` table (no target database)

**Name**
A descriptive name

**Keep statistics**
If not checked then statistics fields (*S_xxx*) will be reseted always when a replication task starts, if checked then the statistics fields will accumulate values. Reset (p.14) them from time to time manually because it's possible non-critical *overrun error* during replication.

**Comment**
Any free comment

## Source database properties

The dialog edits single record of `SCHEMADB` table where `DBID=0`. If scheme type is *record history* then only one group is possible.

**Source database**
Source database, note that *text file* type database is not allowed

**Repl user, password, role**
A user name used by replication server for connecting to the source database. The user must have enough rights to access (SELECT) all the replicated tables (and fields). Changes made by this user are not replicated. If a bidirectional replication should be implemented both schemes (source to target, target to source) must use the same replication user. The password may be encrypted to be human unreadable using * button. Environment macros (p.37) may be used.

**Separator**
An ASCII char value used for delimiting of multi-segment primary keys. Choice a character does not occur in value of any primary key and that is allowed to use in SQL command (not `0`, `EOL`, `EOF`, `'`, etc.). The most suitable are values between `1` and `31`, e.g. `5`. Be patient when transferring offline packages between different code pages (translation to and from UNICODE/UTF-8).

**Disabled**
If checked then replication to all target databases is disabled, the replicator ignores it.

**Comment**
Any free comment

## Target database properties

The dialog edits single record of `SCHEMADB` table where `DBID<>0`. If scheme type is *record history* then no target databases are possible.

**Group Id**
A unique group identifier greater than zero.

**Target database**
Target database, note that both *Interbase* and *text file* types are not allowed

**DB mask**
Database mask uniquely identifies a target database in a database group framework. It's a bit mask. The DBMASK is INTEGER field, therefore max. 31 target databases are allowed in a group (the 32nd bit is sign bit).

**Repl user, password, role**
A user name used by replication server for connecting to the target database. The user must have enough rights to access and modify (SELECT,INSERT,UPDATE,DELETE) all the replicated tables (and fields). Changes made by this user are not replicated. The password may be encrypted to be human unreadable using `*` button. Environment macros (p.37) may be used.

**Disabled**
If checked then replication to the target databases is disabled, the replicator ignores it.

**Separator**
An ASCII char value used for delimiting of multi-segment primary keys. Choice a character does not occur in value of any primary key and that is allowed to use in SQL command (not `0`, `EOL`, `EOF`, `'`, etc.). The most suitable are values between `1` and `31`, e.g. `5`. Be patient when transferring offline packages between different code pages (translation to and from UNICODE/UTF-8).

**Comment**
Any free comment

**Alternate database**
Alternate database containing only replication objects that is used only for offline replication to *text file* database.

**Transfer library**
A transfer library (p.44) to be used for an offline transfer (`transfer_` prefix is excluded).

**Encoder(s)**
Encoder libraries (p.48) to be used for offline package encoding (`enc_` is excluded). If more libraries separated by semicolon (';') is defined then output of the first encoder is passed to the 2nd encoder etc. The name may be followed optionally by slash and subtype name. Every encoder adds an envelope to the offline package therefore decoder knows what library must use to decode it.
Format:

```
libname1[/subtype];libname2[/subtype];...
```

Examples:

```
cmdline/zip
cmdline/zip;cmdline/pgp
```

**Parameters**

Defines runtime parameters for transfer and encoder libraries. Because all values are prefixed by its library name, parameters for all libraries may be held in one field (does not matter if they are active or inactive).
The parameters are written in `<name>=<value>` format. Every parameter resides an individual line. The parameter names are case insensitive. The parameter name may be post fixed by `.0`(such parameter is used by source replicator) or `.1`(such parameter is used by target replicator). It enables using of the same configuration database at both sides.
Lines starting with semicolon (';'), sharp ('#') or slash ('/') are comments.
Environment macros (p.37) may be used. Any value may be encrypted to be human unreadable using * button. Select the value right to equal sign and press * button.
Parameter is looked for in following sequence:

1. `<library>.param.{1|0}`
2. `<library>.param`
3. default value

Example:

```
netdir.dropoffdir=$(LOGPATH)/shared/$(SCHEMAID)-$(DBID)

ftp.host=replicator.my-domain.com
ftp.username=repl
ftp.password=****
ftp.dropoffdir=/home/shared/$(SCHEMAID)-$(DBID)
ftp.passivemode=0

email.address.0=ibrepl1@my-domain.com
email.address.1=ibrepl2@my-domain.com
email.pop3.host=pop3.my-domain.com
email.pop3.port=110
email.pop3.username.0=ibrepl1@my-domain.com
email.pop3.username.1=ibrepl2@my-domain.com
email.pop3.password=****
email.smtp.host=smtp.my-domain.com
email.smtp.port=25
```

## Relation properties

The dialog edits single record of `RELATIONS` table.

**Index**

Relation identifier greater than zero that determines uniquely relation index (position) in a group. The index may be changed safely, remaining relations are renumbered automatically.

**Source relation**

Name of a source table or view.

**Target relation**

Name of a target table/stored procedure where changes are written to.

**Target type**

- *table* - target is a table

- *stored procedure* - target is a stored procedure

In the stored procedure is expected mandatory input parameter `VAR$OPER` (probably of `CHAR(1)` type) where is passed kind of operation *I*..insert, *U*..update, *D*..delete.

**Disabled**
If checked then no replication triggers are generated for the relation.

**Wipe log when deleted**
If checked and if a record in source database is deleted then all previous replication entries concerning to the record are also deleted. Useful for tables which records have short lifetime period - records are deleted before replication processes record (INSERT,DELETE,replicate). In this case replicator cannot read current record (because does not exist) which is important for fields of type #3 (common fields). Second disadvantage is that replication of obsolete records decreases performance.

**Keep statistics**
If not checked then statistics fields (*S_xxx*) will be reset always when a replication task starts, if checked then the statistics fields will accumulate values. Reset (p.14) them from time to time manually because it's possible non-critical *overrun error* during replication.

**Condition**
Defines filter not to replicate all rows. See conditional replication (p.49)

**Sync order**
Defines order that are relations synchronized in. If does exist a table (B) that depends on a table (A), the table (A) should be synchronized before the table (B). `SYNCORDER` of relation assigned to the table (A) must have lower value that `SYNCORDER` of table (B). It's important to keep referential integrity (constraints).

**Sync actions**
Defines list of synchronization actions, see Sync actions designer (p.22)

**Comment**
Any free comment

**See also:**

Sync action designer (p.22)

## Sync actions designer

The dialog edits `SYNCACTION` field of `RELATIONS` table.
Only online synchronization is supported and a target relation of *table* type is required. Only target database relations are modified, source database is left unchanged.
When synchronization is completed successfully, replication records (`REPL$LOG`, `REPL$MAN`) concerning to scheme should be deleted, see Clean source data (p.12) command.
Analogous to conditional replication is possible setup conditional synchronization to filter rows.

Synchronization engine processes step-by-step all relations according `SYNCACTION` settings. Configuration database can define more actions for each relation. The engine synchronizing according following diagram:

```
I=1
repeat
  for each RELATION ordered by SYNCORDER do
    do sync action at position I  ; SYNCACTION[I]
  I=I+1
until no remaining action;
```

**Database type**
Radio buttons switch mode of visual controls to display settings relate to selected target database type. The actions are equivalent because in one group may be mixed *Interbase* and *text file* target databases.

## Target database is *Interbase*

The engine will establish connection to both source and target databases, do select and comparison of source to target records.
The most conventional combination `'36'`, it means - in the first step go through the source database, insert new and update different records, in the second step go through the target database and delete records which already don't exist in the source database.

**Method**
- *loop source table* - do select of a relation in a source database, loop it and for each record lookup for equivalent record in target database relation
- *loop target table* - do select of a relation in a target database, loop it and for each record lookup for equivalent record in source database relation

**Condition (source/target)**
A condition filtering rows that should be synchronized, references to *Conditional defines*. Specifies a condition that limits rows retrieved to a subset of all available rows to synchronize process. The condition is appended to WHERE clause of source/target relation.

**UPDATE non equal records**
The engine will update non-equal records. Only fields, which the *Do not update field* option is disabled, are affected, see Field conflict options dialog (p.25)

**INSERT new records**
The engine will insert to target database non-existing records. All fields are affected except those with the *Do not insert field* option enabled. Valu of such fields remains `NULL` or is set to default field value. See Field conflict options dialog (p.25)

**DELETE non existing records**
The engine will delete from target database non-existing records in source database.

**Log**
- *direct synchronization* - change target database relation that differs
- *log differential SQL commands* - target database remains unchanged, differential SQL commands are logged (if performing of this script on target database synchronize it)

### Target database is *text file*

The engine will establish connection to source databases and do export of relation to a text file. The text file will contain set of INSERT SQL commands.

**Condition (source)**
SQL WHERE condition filtering rows that should be exported, references to a name defined in *Conditional defines*.

**Log**
- *log to SQL log defined in DATABASES* - the SQL script is written to the file determined as Database name (p.17)
- *log to DB log* - the SQL script is written to the terminal log window and to the DBLog

**Conditional defines**
The condition is written using common WHERE syntax. Note that table has implicit tag *'T'*.
Note that Interbase hidden field `RDB$DB_KEY` may be used. The field uniquely identifies any row in relation. Its value is valid in one transaction. For more information search *RDB$KEY* using a web search engine.
Conditional are defined as list of values:

```
<name1>=<condition1>
<name2>=<condition2>
```

The *Condition (source/target)* item references using `name` to it. Environment macros (p.37) (*SCHEMAID,GROUPID,DBID,DB2ID,RELATIONID*) may be used.
A SQL command is constructed as:

```
SELECT * FROM <table> T WHERE <condition>
```

Example:

```
t.EMP_NO in (select e.EMP_NO from EMPLOYEE e inner join DEPARTMENT d on e.DEPT_NO=d.DEPT_N
```

is expanded

```
SELECT * FROM SALARY_HISTORY T WHERE t.EMP_NO in (select e.EMP_NO from EMPLOYEE e inner jo
```

**See also:**

Relation properties (p.21)

# Field properties

The dialog edits single record of `FIELDS` table.

**Id**
Relation identifier greater than zero that determines uniquely field index (position) in a relation. The index may be changed safely, remaining fields are renumbered automatically.

**Source field**
Field name in source table/view.

**Target field**
Field name in target table/procedure.

**Field type**
1. primary key - field is part of primary key
2. foreign key/required field - field is a foreign key or required field
3. common field - common field

If record is changed, current (NEW) and previous (OLD) values of all fields of type `1` and `2` are stored to replication log, field of #1 to `REPL$LOG`, #2 to `REPL$FIELDS`. Field #3 are not logged anyway.

**Conflict options**
Extended field settings, see Field conflict options (p.25)

## Field conflict options

The dialog edits `OPTIONS` field of `FIELDS` table. The Ext.conflict check (p.33) must be enabled to process conflict checking.

**Do not assign for insert**
When a new record is inserted to target database then do not assign field value. The value remains `NULL` or set to default value.
If target database is *text file* field does not appear in INSERT SQL command.

**Do not update field**
When a record is updated to target database then do not modify field value. The value remains unchanged.
If target database is *text file* field does not appear in UPDATE SQL command.

### UPDATE conflicts

Conflict occurs if replication engine does not find expected value of a target field. It means that target record has been changed since last replication. The engine cannot make decision what value is correct. The conflict options defines rules how to treat such states.

**update only when source field changed**
If checked then are replicated fields only if its value was changed. It limits conflict possibility significantly.

**leave target field value**
If a field conflict occurs then target field remains unchanged. No conflict is reported.
If not checked then source value overrides target value even conflict occurred.

**do record update and report field conflict**
If a field conflict occurs then target **field** remains unchanged and field conflict is reported. Such conflict does not block record update.

**do not record update and report conflict**

If a field conflict occurs then target **record** is not updated and record conflict is reported. So one conflicting field can block record update.

## DELETE conflicts

**do not delete record**

If a field conflict occurs then target record is not deleted and conflict is reported.

**See also:**

Field properties (p.24)

## Examples:

Initial record value:

```
CUST_NO  CUSTOMER        CONTACT_FIRST   CONTACT_LAST  PHONE_NO  ADDRESS_LINE     CITY
1004     Central Bank    E.              Brocket       612110000 66 Lloyd Street  Manchest
```

is changed in source database to

```
CUST_NO  CUSTOMER        CONTACT_FIRST   CONTACT_LAST  PHONE_NO  ADDRESS_LINE     CITY
1004     Central Bank    Elizabeth       Brocket       612119988 66 Lloyd Street  Manchest
```

and in target database to

```
CUST_NO  CUSTOMER        CONTACT_FIRST   CONTACT_LAST  PHONE_NO  ADDRESS_LINE     CITY
1004     Central Bank&Co E.              Brocket       445577444 4 Chapel Street  Liverpoo
```

**Case #1**

Configuration:

```
FIELD          Update if changed..  Leave target value... Do record update+report  Cancel
CUSTOMER       FALSE                FALSE                 FALSE                    FALSE
CONTACT_FIRST  FALSE                FALSE                 FALSE                    FALSE
CONTACT_LAST   FALSE                FALSE                 FALSE                    FALSE
PHONE_NO       FALSE                FALSE                 FALSE                    FALSE
ADDRESS_LINE   FALSE                FALSE                 FALSE                    FALSE
CITY           FALSE                FALSE                 FALSE                    FALSE
ZIP            FALSE                FALSE                 FALSE                    FALSE
COUNTRY        FALSE                FALSE                 FALSE                    FALSE
```

Target database after replication:

```
CUST_NO  CUSTOMER        CONTACT_FIRST   CONTACT_LAST  PHONE_NO  ADDRESS_LINE     CITY
1004     Central Bank    Elizabeth       Brocket       612119988 66 Lloyd Street  Manchest
```

All changes made in target database are lost.

**Case #2**

Configuration:

```
FIELD          Update if changed..  Leave target value... Do record update+report  Cancel
CUSTOMER       FALSE                ANY                   TRUE                     FALSE
CONTACT_FIRST  TRUE                 FALSE                 FALSE                    FALSE
CONTACT_LAST   TRUE                 FALSE                 FALSE                    FALSE
PHONE_NO       TRUE                 TRUE                  FALSE                    FALSE
ADDRESS_LINE   TRUE                 FALSE                 FALSE                    FALSE
CITY           TRUE                 FALSE                 FALSE                    FALSE
ZIP            TRUE                 FALSE                 FALSE                    FALSE
COUNTRY        TRUE                 FALSE                 FALSE                    FALSE
```

Target database after replication:

```
CUST_NO  CUSTOMER        CONTACT_FIRST   CONTACT_LAST  PHONE_NO  ADDRESS_LINE     CITY
1004     Central Bank    Elizabeth       Brocket       445577444 4 Chapel Street  Liverpoo
Conflict of CUSTOMER field is reported
```

Target database record is probably in inconsistent state.

**Case #3**

Configuration:

```
FIELD          Update if changed..  Leave target value... Do record update+report  Cancel
CUSTOMER       ANY                  ANY                   TRUE                     TRUE
```

Target database after replication:

```
CUST_NO  CUSTOMER        CONTACT_FIRST   CONTACT_LAST  PHONE_NO  ADDRESS_LINE     CITY
1004     Central Bank    Elizabeth       Brocket       612119988 66 Lloyd Street  Manchest
Conflict of CUSTOMER field is reported
```

Target record is left unchanged and conflict reported

## Interbase Replication Server (classic)

A replication server (`IBREPLSERVER.EXE`) controls replication process, for each task is waiting in background for a starting alert. The alerts are the manual request, the timer event or the Interbase event. When alerted, the server opens a configuration database and according its settings replicates data from a source to a target database.
The server is running with a visual GUI. There is also an alternate replication server (p.38) running in background, as a console application or as a NT service.
The settings is stored in an INI file `IBREPL.INI`. The replication server periodically checks the INI file. If a change in the INI file is detected the replication server rereads it. This enables on-the-fly INI file modification, e.g. the INI file change forces switching to replicate in opposite direction. The default INI file name may be changed using the `/I:config_filename` command line parameter (p.28). The INI file is not checked when Disable timers&events (p.36) is activated.
When exiting the program instance and a *Shift* key is pressed then the current window position is saved to a `IBREPL.DSK` file. The server reads the `IBREPL.DSK` file when starting, so the replication server opens in the same state (except if the *Shift* key if held pressed). The default file name may be changed using the `/SI:config_filename` command line parameter (p.28).
**See also:**
Settings (p.29)
Task settings (p.33)
Environment (p.37)
Command line (p.28)
Logs (p.35)
Menu commands (p.36)

## Replication Server - Command line parameters

A command line options can override default behavior. The options have higher priority than an INI file options.

**/I:config_filename**
Overrides the default INI file (`IBREPL.INI` in a program directory).

**/MIN**
Run program minimized, overrides a Run minimized (p.30) settings.

**/DISABLED**
Disable automatic replication (p.36), overrides a Disable timers&event for all tasks (p.31) settings.

**/NOSET**
Disables the position saving and restoring anyway.

**/SI:<filename>**
A file name where is saved (restored from) window position, the default file name - `IBREPL.DSK` in a program directory.

**/E:<name>=<value>**
Defines a key for an environment (p.37)

**/TRACESQL**
Enables tracing/profiling of SQL commands using a SQL monitor

## Replication Server - Settings

**Settings** (p.29): Config.database (p.29) Appearance (p.30) Log (p.30) Replication (p.31) Scheduler (p.31)

Manages a global server configuration and default keys for particular task keys. Such keys have labels indicated by blue color. Their value may be overridden in a task configuration (p.33).
In a bottom section is indicated current INI file.
**See also:**
Environment (p.37)
Task settings (p.33)

## Settings / Configuration database

**Settings** (p.29): Config.database (p.29) Appearance (p.30) Log (p.30) Replication (p.31) Scheduler (p.31)

**Database file name**
A name of the configuration database entered in Interbase convention:

| | |
|---|---|
| TCP/IP | `server_name:path/filename` |
| NetBEUI | `\\server_name\path\filename` |
| SPX | `servername@path/filename` |

If a multi-threading replication is used with a local Interbase server, use *"localhost:/path/database.gbd"* syntax. It's limitation of an Interbase client. For more information see Jeff Overcash' messages in an IBX newsgroup. Environment macros (p.37) may be used.
For example:

```
SERVER1:C:/DATA/DATABASE.GDB
localhost:$(_DIR_)/DATA/DATABASE.GDB
```

**SQL dialect**
A dialect used for connecting to a config database. If the SQL dialect is 3 then database object names are case sensitive and can contain spaces.

**User name, Password**
A user name and password that the replication server uses to login in the configuration database. The user must have granted enough rights, to read (SELECT) all the tables and modify (UPDATE) statistic fields (*S_xxx*). The password may be encrypted to be human unreadable using * button. Environment macros (p.37) may be used.

**Object prefix**
A prefix for objects used in a configuration database. Not necessary unless more configuration databases is situated in one *GDB*.

**Def.char set**
A character set of a configuration database

## Settings / Configuration database

**Settings** (p.29): Config.database (p.29) Appearance (p.30) Log (p.30) Replication (p.31) Scheduler (p.31)

**Max.lines in terminal**
Max. lines logged to a terminal window. Too many lines decrease performance significantly.

**Show on systray**
When minimized, show in a systray (hides the server from a taskbar).

**Run minimized**
Run program minimized after start.

**Only one instance**
Allow only one running instance of a server. Use if the server is running from a scheduler. If an instance is already running then the new instance silently terminates.

**Instance id**
Only one server instance of the same *Instance id* may be running when an *Only one instance* options is checked.

**Multi-threading replication**
Enable a multi-threading replication. Replication tasks are running in different threads. If not enabled then the replication tasks are serialized in the main thread.
If enabled then all local server databases must be configured using a *"localhost:/path/database.gdb"* syntax otherwise system crashes.

## Settings / Log

**Settings** (p.29): Config.database (p.29) Appearance (p.30) Log (p.30) Replication (p.31) Scheduler (p.31)

**DB log**
A log where are logged errors and conflicts.

**Repl log**
A log where are logged replication statistics. Generally the *Repl log* should be smaller then the *DB log*.

**File name**
A file where messages are logged. Set `NUL` to disable file logging. Environment macros (p.37) may be used.

**Max.log size, Rotate log count**
If size of a log file exceed *Max.log size* then if a *Rotate log count* is zero/empty the file is deleted else the next rotate log is used. Max.number of possible log files determines *Rotate log count*. Set *Max.log size* to `0` to allow unlimited size of the log file.

**Log error SQL commands**
Log full SQL commands when an error is raised. Its helpful to debug SQL DML (data manipulation language) errors.

**Log error SQL params**
Log SQL command parameter values when an error is raised. Its helpful to resolve referential integrity conflicts.

## Settings / Replication

**Settings** (p.29): Config.database (p.29) Appearance (p.30) Log (p.30) Replication (p.31) Scheduler (p.31)

**Now as UTC**
If key is checked then time stamps are logged as UTC (Universal Time Coordinates)/GMT (Greenwich Mean Time).

**Offline package directory**
Path to a directory where offline packages are stored. The packages are deleted when their acknowledge is received. Environment macros (p.37) may be used. Special `$(OFFDIR)` macro is expanded to *"S"* or *"T"* depending if running at a source or target database side.
Task settings (p.31)

## Settings / Scheduler

**Settings** (p.29): Config.database (p.29) Appearance (p.30) Log (p.30) Replication (p.31) Scheduler (p.31)

**Disable times&events for all tasks**
Disable automatic execution of all tasks globally (both on-timer and on-event). Enables working on a server configuration without unintentional execution of a replication task.
See also command line parameters (p.28)

**Replicate when time matches cron pattern**
Enables a replication when current (local/UTC) time matches a CRON pattern. There is used an extended cron syntax.
The pattern matching engine recognizes the following fields:

| field | meaning | allowed values | note |
|---|---|---|---|
| 1 | minute | 0-59 | |
| 2 | hour | 0-23 | |
| 3 | day of month | 1-31 | |
| 4 | month | 1-12 | or names, see bellow |

| | | | | |
|---|---|---|---|---|
| 5 | day of week | 1-7 (1=Sun, etc.) | or names, see bellow | |
| 6 | year | 100-9999 | optional | |
| 7 | day of year | 1-366, (1=Jan 1th, etc.) | optional | |
| 8 | second | 0-59 | optional | |

A field may be an asterisk (*), which always stands for *first-last*.
Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive. For example, *8-11* for an *hours* entry specifies execution at hours 8, 9, 10 and 11.
Lists are allowed. A list is a set of numbers (or ranges) separated by commas. Examples: *1,2,5,9*, *0-4,8-12*.
Step values can be used in conjunction with ranges. Following a range with */<number>* specifies skips of the number's value through the range. For example, *0-23/2* can be used in the hours field to specify command execution every other hour (the alternative in the V7 standard is *0,2,4,6,8,10,12,14,16,18,20,22*). Steps are also permitted after an asterisk, so if you want to say *every two hours*, just use **/2*.
Names can also be used for the *month* and *day of week* fields. Use the first three letters of the particular day or month (case doesn't matter). Ranges or lists of names are not allowed.
Note: The day of a command's execution can be specified by two/three fields - day of month, day of week and day of year. If both fields are restricted (i.e. aren't *), the command will be run when either field matches the current time. For example, *30 4 1,15 * 5* would cause a command to be run at 4:30am on the 1st and 15th of each month, plus every Friday.
When an optional field is omitted, it is treated as ignore by the expression evaluator as opposed to the '*' which stands for match all. Like the asterisk '*', the under bar '_' also has special meaning and is used to ignore imbedded fields.
Special FreeBSD keywords may be used:

| Keyword | Meaning |
|---|---|
| `@yearly` | *0 0 1 1 ** |
| `@annually` | *0 0 1 1 ** |
| `@monthly` | *0 0 1 * ** |
| `@weekly` | *0 0 * * 0* |
| `@daily` | *0 0 * * ** |
| `@midnight` | *0 0 * * ** |
| `@hourly` | *0 * * * ** |

See UNIX *crontab(5)* manual page for more details.

#### Interval

A minimal interval that must elapse until a next replication is started using cron. Note it have effect only during running instance of a replication server.
Syntax: [*d][*h][*m][*s]
Examples:
- 1d = 1 day
- 1h30m = 90 minutes

#### Replicate on event

If an event `<prefix>REPLICATENOW` occurs the replication server executes a replication process. Changes are transferred to target databases as soon as possible. Name of the "hot" event may be changed (default: `REPL$REPLICATENOW`). Note that Interbase events may decrease performance significantly if the replication server is running on a different machine than the Interbase server - events are transferred over the network.

**Max.repl.runtime**
Max.time that a replication task may be running. If *Max.repl runtime* is elapsed the replication task is interrupted. The same syntax as in *Interval* key is used.
**See also:** Task settings (p.31)

**Do not replicate if file exists**
Disables a replication (or forces stopping it when running) task execution if a defined file (semaphore) does exist. It enables disabling of the replication when an administration task should be performed, for example system backup that need access to Interbase databases exclusively. Environment macros (p.37) may be used.

## Task settings

**Task settings** (p.33): Common (p.33) General (p.33) Offline (p.34) Log (p.34) Scheduler (p.35)

The replication server can control any number of tasks. The task defines method how to replicate, when to replicate. The task is fired either automatically or manually.
Keys indicated by a blue label overrides global settings - if non-empty. Checkboxes are using a tri-state logic.

## Task settings / Common

**Task settings** (p.33): Common (p.33) General (p.33) Offline (p.34) Log (p.34) Scheduler (p.35)

**Task name**
A name that appears in a task list.

**Comment**
Any free comment

## Task settings / General

**Task settings** (p.33): Common (p.33) General (p.33) Offline (p.34) Log (p.34) Scheduler (p.35)

| Type | Replicate | |
|---|---|---|
| online | records | connect to both source and target databases and replicate records |
| online | repl.log | connect to both source and target databases and transfer source `REPL$LOG` to target `REPL$LOG` |
| offline-source | - | connect to source database and prepare/process offline packages |
| offline-target | records | connect to target database and replicate records taken from offline packages |
| offline-target | repl.log | connect to target database and store changes to `REPL$LOG` |
| target | - | connect to a target database and replicate target `REPL$LOG` |
| synchronization | - | connect to both source and target databases and perform synchronization |

**SchemaId(s), GroupId(s), SrcDBId(s), TgtDBId(s)**
Identifies a scheme which task will be processed. Use a ... button to select it from a list provided from a configuration database. Press *Alt+Up/Down* to show lookup dialog.

**Now as UTC**
Overrides global settings (p.31) of an identical name.

**Ext.conflict check**
If is checked then extended conflict checking is performed, it means that is possible to resolve field or record conflicts. The extended conflict checking is slower because more select to a target database must be performed.

**Report to source / target**
Replication errors and conflicts are logged to `REPL$MAN` of a source/target database. Enables to choice which database administrator will be responsible for manual conflict resolving.

## Task settings / Offline

**Task settings** (p.33): Common (p.33) General (p.33) Offline (p.34) Log (p.34) Scheduler (p.35)

**Offline package directory**
Overrides global settings (p.31) of an identical name.

**Receive data**
Receive offline packages prior to a replication (`REPL$LOG` processing)

**Send data**
Send pending offline packages.

**Resend data**
Resend packages if a package (which has been sent after another one is acknowledged but former is still unacknowledged. - probable corruption or lost package).
Example:

| TRANSFERID | STAMP_SENT | TRANSFERID_ACK |
|---|---|---|
| 10 | 1.1.2000 | 2 |
| 11 | 2.1.2000 | NULL |
| 12 | 3.1.2000 | NULL |

Now if an acknowledge of the package #12 come, the package #11 is resented. In normal flow must correspond an order of the packages to the order of the acknowledges. So if the acknowledge of the package #12 has been received, the package #11 was lost - the opposite side did not received it correctly.

## Task settings / Log

**Task settings** (p.33): Common (p.33) General (p.33) Offline (p.34) Log (p.34) Scheduler (p.35)

**DB log, Repl log**
Overrides a DB log, Repl log global settings (p.30).

**Log name**
An identifier that is logged in every message. It enables to distinguish messages if more tasks are logging to a shared file.

## Task settings / Scheduler

**Task settings** (p.33): Common (p.33) General (p.33) Offline (p.34) Log (p.34) Scheduler (p.35)

**Disable timer&event**
Disables auto task execution.

**Others**
Overrides global settings (p.31).

## Replication Server - Logs

**Tasks**
A list of tasks that the replication server controls.
**See:** File commands (p.36) Task settings-Log (p.33)

**Replication log, DB log**
Replication process messages are written to a *Replication log*, possible errors/conflict to a *DB log*. The messages are passed through a terminal window to a file.
**See:** Settings-Log (p.30), Task settings-Log (p.34), Max.lines in terminal (p.30), Edit/Clear (p.36)

**Repl.log table**
`REPL$LOG` records considering to a current replication task. The record can be deleted from current scheme using *[-]* button at toolbar (according its `DBMASK`). To take effect acquire manual commit.
A *Fields* tab shows a list of logged foreign fields in `REPL$FIELD`.

**Manual log table**
`REPL$MAN` records considering to current replication task. Record can be deleted using *[-]* button at toolbar or moved back to a `REPL$LOG` using *[+]* button at toolbar (according its `DBMASK`). To take effect acquire manual commit.
A *Fields* tab shows list of logged foreign fields in `REPL$FIELD`.
The *Description* tab shows a reason of an error.
The *Conflict fields* tab indicates values that cause a conflict (in a XML format according `conflict.dtd`).

**Transfer table**
A list of offline transfers for given task. Clear *STAMP_SENT*, *STAMP_REC* or *STAMP_PROC* to reprocess a offline package. To take effect acquire manual commit.

## Replication Server - Menu commands

**File/New task**
Add a new task to a task list
Task settings (p.33).

**File/Edit task**
Edit a current task.
Task settings (p.33).

**File/Delete task**
Delete a current task from a task list.

**File/Move up**
Move a current task in a task list up.

**File/Move down**
Move a current task in a task list down.

**File/Settings**
Show a settings dialog (p.29).

**Edit/Clear**
Clear a replication or a DB log terminal window.

**File/Clear cached settings**
The replication server caches values from a configuration database to improve performance. Use this command if a scheme definition in the configuration database has been changed when the replication server is running to reread values to cache during next execution loop.

**Run/Replicate**
Start a replication of a given task manually.

**Run/Refresh log record count**
Refreshes *#log* and *#man* columns that indicates number of records remaining in the `REPL$LOG`/`REPL$MAN` tables.

**Run/Stop**
Force stopping of a current task.

**Run/Stop all**
Force stopping of all running tasks.

**Run/Disable timers&events**
If checked then disables auto execution of all replication tasks and INI file checking. See also Disable timers&events for all tasks (p.31) and command line (p.28)

**Help/Contents**
Show help contents

**Help/Home page**
Go to replicator's home web site.

**Help/About**
Show program info and release number.

## Environment

The environment is set of variables. The variable identified by a macro name written `$(variable)` is expanded. The `$(variable)` string is substituted is substituted by the variable value taken from the environment. The environment enables to define and distribute only single configuration database - all replication server use the config database containing the same data. It simplifies configuration maintaining.
The environment variable is searched in following sequence:
1. implicit variables (*_VER_*,*_DIR_*,*DBID*,...)
2. variables defined using `/E` command line parameter (p.28)
3. variables defined using *Environment* command, stored in an INI file
4. variables from `REPL$ENVIRONMENT` table in a source/target database
5. variables from `ENVIRONMENT` table in a config database
6. variables from a system environment, keys prefixed *IBREPL_*
7. variables from the system environment

Implicit variables vary when is looked for the variable (what command is called)

| | |
|---|---|
| *_DIR_* | program directory |
| *_VER_* | IBReplicator version |
| *SCHEMAID* | schema id |
| *GROUPID* | groupid |
| *DBID* | database id |
| *DB2ID* | target database in offline replication |
| *OFFDIR* | offline temporary directory |

In `ENVIRONMENT` tables exist following mandatory records

| | |
|---|---|
| *_VERSION* | database version |

To setup the environment entries in environment dialog specify variables in *'name=value'* form. Do not add quotes for long names, for example:

```
DBPATH=C:\Documents and settings\DATA
LOGPATH=$(DBPATH)
USER=sysdba
PASSWORD=masterkey
```

# Interbase Replication Server (service)

A replication server (`IBREPLSERVERSVC.EXE`) controls replication process. It implements the same functionality as replication server (p.28) the only difference is that the *service* server is running in background whereas the *classic* server provides a visual interface.
The service is configured using an INI file. The INI file uses format of the classic replication server (p.28) therefore the best way is to define all task using GUI of the classic server and pass name of the INI file using an `/I` parameter.
One service instance controls all tasks defined in the INI file. If on one machine has to be running more replication servers it's necessary identify the particular server using name (p.38) parameter.
The *service* server supports dual interface, it can run either as a NT service or a console application. The NT service requires Windows NT/2000/XP, the console application is running on any 32-bit Windows platform (Windows 9x/ME/NT/2000/XP).
Any program that is able running as the NT service must be inserted to a list of services. The list is managed by a system service manager. There is 3rd party software that do it. But the *service* server supports self registration using `/INSTALL` parameter. When installed as the NT service, new entry *"IBReplServer service 'default'"* appears in the service list. From here is the service controlled using a properties dialog. In the properties dialog setup if the service is started automatically when the system starting or manually. Also you can stop, pause or continue the service, it's equivalent to command line command.
An example how install 2 NT services and run third console server

```
  ibreplserversvc First /I:IBREPL1.ini /INSTALL
  ibreplserversvc Second /I:IBREPL2.ini /INSTALL
  ibreplserversvc Third /I:IBREPL3.ini
```

Every running server provides its current status. The status is held in a shared FILE_MAPPING memory. To print the status use /STATUS (p.38) parameter.
For example to print status of *Second* instance from previous example:

```
  ibreplserversvc Second /STATUS
```

Note that a multi-threading is supported.

## Command line

The service server is controlled from the command line.

```
Usage: ibreplserversvc [<name>] [/STATUS] [/PAUSE] [/CONTINUE] [/STOP] [/KILL]
                        [/RESET] [/I:<inifile>] [/N:<machine>] [/INSTALL]
                        [/U:<user>] [/P:<psw>] [/F:<file>] [/UNINSTALL] [/SILENT]
                        [/SECURITY] [/O:<stdout>] [/H] [/?]


Description:
  <name>      name of service, enable running more replication servers, default: "default"
  <inifile>   configuration file, default IBREPL.INI in program directory

  /PAUSE      stop all running tasks, pause the replication server
  /CONTINUE   continue (unpause)
  /STOP       stop replication server
  /KILL       kill replication server, not recommended

  /STATUS     print current status
  /RESET      reset status

  /INSTALL    install NT service
    <user>    service account "DomainName\Username", default: "LocalSystem"
    <psw>     service account password
    <machine> NT service machine name
    <file>    file where is logged current status (FILE_MAPPING)
  /UNINSTALL  uninstall NT service, remove from NT service list

  /SILENT     no stdout output
  <stdout>    file used for stdout
  /SECURITY   print security info

  /? /H       print help info
```

## IBREPLC.EXE

A command line replicator tool that can replicate a replication task. The tool is running as a console application and no user control is required. The main difference between a service replicator (p.38) and the `IBREPLC.EXE` is that the service tool is running until is manually terminated whereas this tool is executed, process a required task and terminates by itself.
The tools is executed manually either from the command line, from other software or from a scheduler. In the scheduler is possible to setup exact time when to replicate.
All the functionality of the replication server (p.28)is supported. See command line help `IBREPLC.EXE /?` for a list of parameters.
Using the `/I:<ini>` parameter set an INI file location. All the INI file parameters may by overridden at the command line.

## IBREPLINST.EXE

A command line replicator tool that can manage features managed from the replication manager (p.8). The tool is running as a console application and no user control is required. Using this tool is possible to create a custom batch file that will be responsible for replication environment installation.
See command line help `IBREPLINST.EXE /?` for list of parameters. Corresponding features are explained in the replication manager (p.8).

## IBREPLSCR.EXE

A command line SQL scripting tool that processes a SQL script and executes SQL commands to a source or target database. Special tags may be placed to the SQL script. Because these tags are comments they have no effect at a common SQL exec tools (`ibconsole`, `ibsql`...). But the `ibreplscr` recognizes them and enables select processing.
See command line help `IBREPLSCR.EXE /?` for list of parameters.
Using an `/I:<ini>` parameter set an INI file location or set all parameters from command line. The command line parameters override the INI file settings.

## The tags used in the SQL script

`/* @YYYYMMDD hhmmss */`
a stamp when the replication server made a record (the stamp of first record in a replication loop)
`/* #SEQID */`
Current SEQID from REPL$LOG.
`/* BLOB_TXT:table.field where<CRLF>blob content*/`
Because is impossible change a BLOB field using a plain SQL command, values of all not-NULL BLOB fields are written to the SQL script.
`Table.field` identifies the BLOB field (column), `where` identifies the record (row). `Blob content` follows, note that "*/" sequence is changed to "*_/" and backslashes are doubled.
`/* BLOB_BIN:table.field where<CRLF>hex blob content*/`
The same as in text BLOB but data are encoded in hex string

## IBREPLPACKAGE2XML.EXE

A command line tool that exports content of in offline binary package to a XML file. The XML is formatted according `offpackage.dtd`.
The exported file may help to resolve a replication problem or can be used to feed the package to own database.
The tool is able to export the packages stored in a temporary offline directory or the shared packages that are enveloped.

## Transfer offline libraries

Replicator sends and receives packages to/from a drop-off location using a transfer library. The transfer library is a common dynamic loadable library that is loaded and released during replication process (dynamic linking).

## transfer_email

The library exchanges offline packages using common email communication. Offline packages are sent to a SMTP server and read from a POP3 server. Using of a mailbox that is dedicated to one scheme is recommended.

### Configuration parameters (p.19)

**email.SMTP.Host[.{0|1}]=<host name>**
Host name used for outgoing emails.
*Example:* `smtp.email.com` or `192.168.1.1`

**email.SMTP.Port[.{0|1}]=<port>**
Port where SMTP server listening
*Default:* `25`

**email.SMTP.AuthType[.{0|1}]=<auth>**

| | |
|---|---|
| `0` | no authentication is required, default |
| `1` | simple authentication is required (AUTH LOGIN) |

**email.SMTP.UserName[.{0|1}]=<user>**
Login user name if required by SMTP

**email.SMTP.Password[.{0|1}]=<psw>**
Authorization password if authentication required

**email.SMTP.ReadTimeout[.{0|1}]=<timeout>**
ReadTimeout is an integer property that indicates the number of milliseconds that the connection should wait for the peer connection to become readable using the protocol stack.
*Default value:* `20000 ms` (`0`=infinity)

**email.Address.0=<address>**
Sender address (From).

**email.Address.1=<address>**
Addressee (To).

**email.POP3.Host[.{0|1}]=<host>**
Host name of POP3 server used for incoming emails.

**email.POP3.Port[.{0|1}]=<port>**
Port where SMTP server listening
*Default:* `110`

**email.POP3.APOP[.{0|1}]=0**
`1` use the alternate authentication method providing secure password exchange over the network

**email.POP3.UserName[.{0|1}]=<user>**
Username that represents the account identity used for the POP3 account

**email.POP3.Password[.{0|1}]=<psw>**
Password that represents the password used for the POP3 account.

**email.POP3.ReadTimeout[.{0|1}]=<timeout>**
ReadTimeout is an Integer property that indicates the number of milliseconds that the connection should wait for the peer connection to become readable using the protocol stack.
*Default value:* `20000 ms` (`0`=infinity)

**email.tempdir[.{0|1}]=<temp dir>**
Directory where are created temporary files.
*Default value:* system temporary directory

### Example:

```
; email of source side replicator
email.address.0=ibrepl1@my-domain.com
; email of target side replicator
email.address.1=ibrepl2@my-domain.com
; POP3 server, the same for both side replicators
email.pop3.host=pop3.my-domain.com
email.pop3.port=110
; authorization
email.pop3.username.0=ibrepl1@my-domain.com
email.pop3.username.1=ibrepl2@my-domain.com
email.pop3.password=****
; outgoing server
email.smtp.host=smtp.my-domain.com
email.smtp.port=25
```

## transfer_ftp

The library exchanges offline packages using common FTP protocol. A FTP server (visible from both ends) must exist.

### Configuration parameters (p.19)

**ftp.Host[.{0|1}]=<host>**
The IP address or host name for the computer acting as a FTP server connections.

**ftp.Port[.{0|1}]=<port>**
Represents the port number used for the FTP Control channel.

**ftp.Passive[.{0|1}]={0|1}**
Determines how a File Transfer Protocol Data connection is made.

| | |
|---|---|
| 0 | the PORT method is used |
| 1 | the PASV method is used |

**ftp.UserName[.{0|1}]=<user>**
The user authorization provided for authentication of client connections to an FTP server.

**ftp.Password[.{0|1}]=<psw>**
The user authorization provided for authentication of client connections to an FTP server.

**ftp.ReadTimeout[.{0|1}]=<timeout>**
ReadTimeout is an Integer property that indicates the number of milliseconds that the connection should wait for the peer connection to become readable using the protocol stack.
*Default value:* `20000 ms` (`0`=infinity)

**ftp.Proxy.Host[.{0|1}]=<host>**
The IP address or host name for the computer acting as a proxy server for FTP client connections.

**ftp.Proxy.Port[.{0|1}]=<port>**
The port number used for connections to the computer acting as a proxy server for FTP client connections.

**ftp.Proxy.UserName[.{0|1}]=<user>**
The user authorization provided for authentication of client connections to an FTP proxy server.

**ftp.Proxy.Password[.{0|1}]=<psw>**
The user authorization provided for authentication of client connections to an FTP proxy server.

**ftp.Proxy.Type[.{0|1}]=<type>**
Connection and authentication methods permitted for proxy connections to an FTP server.
`0` NONE sends the FTP client Username and optional Password.
`1` USERSITE sends the ProxySettings Username and optional Password, followed immediately by the FTP client Username and optional Password.
`2` USER sends the ProxySettings Username and optional Password, sends the SITE command and the FTP Host property, and the FTP client Username and optional Password.
`3` OPEN sends the ProxySettings Username and optional Password, sends the OPEN command with the FTP Host property, and the FTP client Username and optional Password.
`4` USERPASS calls SendCmd with the command USER user@proxyuser@host, calls SendCmd with the command PASS pwd@proxypwd, or optionally PASS pwd.
`5` TRANSPARENT sends the ProxySettings Username and optional Password, followed immediately by the FTP client Username and optional password.
`6` HTTPPROXYWITHFTP not implemented at this time.

**ftp.DropOffDir[.{0|1}]=<dir>**
The FTP server directory acting as shared drop-off directory for source and target replicators.

**Example:**

```
ftp.host=replicator.my-domain.com
ftp.username=repl
ftp.password=****
ftp.dropoffdir=/home/shared/$(SCHEMAID)-$(DBID)
ftp.passivemode=0
```

## transfer_netdir

The library exchanges offline packages to/from a net folder. This encoder is helpful for testing and debugging.

### Configuration parameters (p.19)

**netdir.dropoffdir[.{0|1}]=<directory>**
The shared directory acting as shared drop-off dir for source and target replicators.

**Example:**

```
netdir.dropoffdir=$(LOGPATH)/shared/$(SCHEMAID)-$(DBID)
```

# Encoder offline libraries

Using an encoder library is possible process offline package (before is sent and after has been received). You can for example improve security if you encrypt offline packages using tripple-DES or you can save your bandwidth if big packages are packed (ZIP, ARJ).

# enc_cmdline

The encoder saves offline package to a file in temporary directory, passes name of the file to a command line driven utility. The utility output is enveloped and passed to next encoder or to transfer library. Analogous decoding is performed.
The encoder library may process more tasks. The required task is specified in encoder list by a subtype name that follows the encoder name, for example `cmdline/zip`.

## Configuration parameters (p.19)

**cmdline[_<subtype>].tempdir=<temp dir>**
Directory where are created temporary files
*Default value:* system temporary directory

**cmdline[_<subtype>].cmd.0=<command line>**
Command line for decoding.
*Default values:*
```
 cmdline_zip.cmd.0=pkunzip "%src%" "%tmpdir%"
 cmdline_arj.cmd.0=arj e "%src%" "%tmpdir%"
```

**cmdline[_<subtype>].cmd.1=<command line>**
Command line for encoding.
*Default values:*
```
 cmdline_zip.cmd.1=pkzip "%dest%" "%src%"
 cmdline_arj.cmd.1=arj a "%dest%" "%src%"
```
**Parse key words for *cmdline* parameters:**

| | |
|---|---|
| `%src%` | source file name |
| `%dest%` | target file name |
| `%name%` | offline package name |
| `%tmpdir2%` | temporary dir with last slash |
| `%tmpdir%` | temporary dir without last slash |

# Conditional replication

Enabled run-time conditional replicating - if record should be replicated. Text of this field is inserted into AFTER INSERT/UPDATE/DELETE triggers that are called when record has been changed. If condition you can affect if write it to REPL$LOG. There are declared some variables in every trigger.
Template of a generated trigger:

```
CREATE TRIGGER
AFTER UPDATE|INSERT|DELETE
MANDATORY VARIABLE DECLARATION
FACULTATIVE VARIABLE DECLARATION
MANDATORY PART
CONDITIONAL PART
IF (VAR$COND > 0) THEN
  INSERT INTO REPL\$LOG
```

Condition field:

```
FACULTATIVE VARIABLE DECLARATION
CONDITIONAL PART
```

**Mandatory variables**

- **`VAR$SEPARATOR CHAR(1)`**

Separator used as primary key delimiter

- **`VAR$OLDPKEY VARCHAR(MaxKeyLength (p.16))`**

Primary key values valid before update/delete

- **`VAR$NEWPKEY VARCHAR(MaxKeyLength (p.16))`**

Primary key values valid after update/insert

- **`VAR$ID INTEGER`**
- **`VAR$COND INTEGER`**

The most important variable. Reset (to `NULL` or `0`) the variable to not to write record to `REPL$LOG` or `REPL$HISTORY`.

- **`VAR$SCHEMAID INTEGER`**, **`VAR$RELATIONID INTEGER`**, **`VAR$GROUPID INTEGER`**, **`VAR$DBMASK INTEGER`**

Identifies schema its record is replicated, `DBMASK` is not used in record history type.

- **`VAR$RELATIONNAME VARCHAR(100)`**

Name of relation its record is replicated

- **`VAR$OPER CHAR(1)`**

Kind of operation (`D`..delete, `I`..insert, `U`..update)

- **`VAR$REPLUSER CHAR(31)`**

Defined replication user for current scheme.

**Facultative variables**

Any variable declaration, from top of field while line beginning `DECLARE VARIABLE`.

**Conditional part**

Aim is setting or resetting of `VAR$OPER` variable. It is possible to use any relation fields using `NEW`/`OLD` prefix. But because part is common for all kinds of triggers (INSERT/DELETE/UPDATE) and in the INSERT trigger is illegal `OLD` and in the DELETE trigger is illegal `NEW` prefix, new parse symbols are defined.

- **`{#CXO#}`** .. substitute `OLD` if possible (UPDATE/DELETE), otherwise `NEW` (INSERT)

- **`{#CXN#}`** .. substitute `NEW` if possible (UPDATE/INSERT), otherwise `OLD` (DELETE)
- **`{#IF <cond1>,<cond2>,...#}`**
  Following code (until next `{#IF#}` symbol) is used or omitted according to `<cond>`
  - ALL .. always true
  - INSERT,UPDATE,DELETE .. insert if kind of trigger in the list
  - !INSERT,!UPDATE,!DELETE .. insert unless kind of trigger in the list

**`REPL$CONFIG`**
The table contains data that are required in runtime in replication triggers. The table is fed by Replication manager (p.8). A administrator need not take care of it, with one exception.
There is the field `DISABLED` that enables to switch-off logging to `REPL$LOG` or `RELP$HISTORY` tables. To disable logging for particular schemes set value to *'N'*.

**Example**

```
DECLARE VARIABLE VAR$PLACEID INTEGER;
DECLARE VARIABLE VAR$PLACEID_A INTEGER;

SELECT PLACEID_A FROM TASKS    /* list of tasks assigned to distributed places */
WHERE ID={#CXO#}.TASKID
INTO VAR$PLACEID_A;

SELECT ID FROM PLACE           /* list of places */
WHERE SCHEMAID=:VAR$SCHEMAID
INTO VAR$PLACEID;

{#IF DELETE#}
                               /* insert if trigger is delete */
{#IF ALL#}
IF (VAR$PLACEID IS NULL OR VAR$PLACEID_A<>VAR$PLACEID) THEN
  VAR$COND = 0;               /* do not replicate "foreign" task */
```

# Unit IBDataSet2

Auxiliary unit that improves efficiency of `TIBDataSet`

## TIBDataSet2

```
type
  TIBDataSet2 = class( TIBDataSet );
```

Overrides `TIBDataSet` and improves efficiency - limits obsolete SQL commands that break `TIBReplicator` (p.62) flow.

### QSelect (TIBDataSet2)

```
public
  property QSelect;
```

### QDelete (TIBDataSet2)

```
public
  property QDelete;
```

### QModify (TIBDataSet2)

```
public
  property QModify;
```

### QInsert (TIBDataSet2)

```
public
  property QInsert;
```

### PrepareBlobs (TIBDataSet2)

```
public
  procedure PrepareBlobs;
```

### SetFieldAsVariant (TIBDataSet2)

```
public
  procedure SetFieldAsVariant(
      Field: TField;
      const Val: Variant );
```

Fixes TIBCustomDataSet.InternalSetFieldData where empty string in a TIBStringField is always changed to Null

## Register

```
procedure Register;
```

Used by IDE to register component at palette

# Unit IBReplicator

A developer library that offers all IB Replicator Suite functionality to independent developers. It's possible implement all functionality into a custom application and no external executables are necessary anymore.

## TValueList

```
type
  TValueList = class( TStringList );
```

Sorted `TStringList` that implements `Values` (p.52) property. TStringList disables writing to Values when `Sorted` is `True`.

### Values (TValueList)

```
public
  property Values[ const Name: string ]: string read write;
```

just overrides `TStrings.Values`

## TReplFieldDefs

```
type
  TReplFieldDefs = class;
```

Structure for caching config database parameters

**See also**
`TReplLogRecord` (p.55)

### ClearTags (TReplFieldDefs)

```
public
  procedure ClearTags( aTagMask: Integer );
```

### BuildFieldDefs (TReplFieldDefs)

```
public
  procedure BuildFieldDefs(
      const CN: string;
      fCachedList: TStrings;
      aSQLDialect_Src: Integer;
      aSQLDialect_Tgt: Integer;
      IUD: Char );
```

## GetWhereParams (TReplFieldDefs)

```
public
  procedure GetWhereParams(
      aSQLDialect_Src: Integer;
      aSQLDialect_Tgt: Integer;
      const aPrefix_Tgt: string;
      var aWhere_Src: string;
      var aWhere_Tgt: string );
```

## GetWhereValues (TReplFieldDefs)

```
public
  procedure GetWhereValues(
      aSQLDialect_Src: Integer;
      aSQLDialect_Tgt: Integer;
      IUD: Char;
      const aOldPKey: TStringOpenArray (p.89);
      const aNewPKey: TStringOpenArray (p.89);
      var aWhere_Src: string;
      var aWhere_Tgt: string );
```

## Destroy (TReplFieldDefs)

```
public
  destructor Destroy; override;
```

## Fields (TReplFieldDefs)

```
public
  Fields: array[ 1 .. 3 ] of array of TReplFieldItem (p.98);
```

## FieldRefs (TReplFieldDefs)

```
public
  FieldRefs: array of TReplFieldRef (p.98);
```

## RelationName_Src (TReplFieldDefs)

```
public
  RelationName_Src: string;
```

## RelationName_Tgt (TReplFieldDefs)

```
public
  RelationName_Tgt: string;
```

## Type_Tgt (TReplFieldDefs)

```
public
  Type_Tgt: Char;
```

## Where_Src (TReplFieldDefs)

```
public
  Where_Src: string;
```

### Where_Tgt (TReplFieldDefs)

```
public
  Where_Tgt: string;
```

### WhereParam_Src (TReplFieldDefs)

```
public
  WhereParam_Src: string;
```

### WhereParam_Tgt (TReplFieldDefs)

```
public
  WhereParam_Tgt: string;
```

### InsertSQL_1 (TReplFieldDefs)

```
public
  InsertSQL_1: string;
```

### InsertSQL_2 (TReplFieldDefs)

```
public
  InsertSQL_2: string;
```

### ModifySQL (TReplFieldDefs)

```
public
  ModifySQL: string;
```

## TReplLogField

```
type
  TReplLogField = class;
```

Abstraction of a source database field

**See also**
`TReplLogRecord` (p.55) `TReplLogField_IBSQL` (p.60) `TReplLogField_Offline` (p.60)

### DataType (TReplLogField)

```
public
  property DataType: TFieldType read;
```

**See also**
`TFieldType`

### DataType_F (TReplLogField)

```
public
  property DataType_F: Char read;
```

**See also**
`TOffReplRecValue` (p.92).DataType

### asString (TReplLogField)

```
public
  property asString: string read;
```

### asVariant (TReplLogField)

```
public
  property asVariant: Variant read;
```

### asInteger (TReplLogField)

```
public
  property asInteger: LongInt read;
```

### asFloat (TReplLogField)

```
public
  property asFloat: Extended read;
```

### asInt64 (TReplLogField)

```
public
  property asInt64: Int64 read;
```

### asDateTime (TReplLogField)

```
public
  property asDateTime: TDateTime read;
```

### IsNull (TReplLogField)

```
public
  function IsNull{}: Boolean; virtual; abstract;
```

### IsBlob (TReplLogField)

```
public
  function IsBlob{}: Boolean;
```

## TReplLogRecord

```
type
  TReplLogRecord = class;
```

Abstraction of a source database `REPL$LOG` record

**See also**

`TReplLogField` (p.54) `TReplTargetDatabase` (p.57).`ReplicateRecord` (p.58) `TReplLogRecord_Online` (p.61) `TReplLogRecord_Offline` (p.62)

### Src_Record (TReplLogRecord)

```
public
  property Src_Record[ Idx: Integer ]: TReplLogField (p.54) read;
```

### Src_IsEmpty (TReplLogRecord)

```
public
  property Src_IsEmpty: Boolean read;
```

### Src_RecordCount (TReplLogRecord)

```
public
  function Src_RecordCount{}: Integer;
```

### Src_FieldExists (TReplLogRecord)

```
public
  function Src_FieldExists( Idx: Integer ): Boolean;
```

### FieldQ_Locate (TReplLogRecord)

```
public
  function FieldQ_Locate(
      aNew: Char;
      aIdx: Integer ): TReplLogField (p.54); virtual; abstract;
```

### FieldQ_IsEmpty (TReplLogRecord)

```
public
  function FieldQ_IsEmpty{}: Boolean; virtual; abstract;
```

### Create (TReplLogRecord)

```
public
  constructor Create( aReplicator: TIBReplicator (p.62) );
```

### Destroy (TReplLogRecord)

```
public
  destructor Destroy; override;
```

### SchemaId (TReplLogRecord)

```
public
  SchemaId: Integer;
```

### GroupId (TReplLogRecord)

```
public
  GroupId: Integer;
```

### SeqId (TReplLogRecord)

```
public
  SeqId: Integer;
```

### RelationId (TReplLogRecord)

```
public
  RelationId: Integer;
```

### CN (TReplLogRecord)

```
public
  CN: string;
```

### IUD (TReplLogRecord)

```
public
  IUD: Char;
```

### OldPKey (TReplLogRecord)

```
public
  OldPKey: TStringOpenArray (p.89);
```

### NewPKey (TReplLogRecord)

```
public
  NewPKey: TStringOpenArray (p.89);
```

### NewFKey (TReplLogRecord)

```
public
  NewFKey: TStringOpenArray (p.89);
```

### Separator (TReplLogRecord)

```
public
  Separator: Char;
```

### Stamp (TReplLogRecord)

```
public
  Stamp: TDateTime;
```

### FieldDefs (TReplLogRecord)

```
public
  FieldDefs: TReplFieldDefs (p.52);
```

## TReplTargetDatabase

```
type
  TReplTargetDatabase = class;
```

Abstraction of a target database

**See also**
TReplTargetDatabase_IB (p.59) TReplTargetDatabase_Logger (p.60)

### Replicator (TReplTargetDatabase)

```
public
  property Replicator: TIBReplicator (p.62) read;
```

parent replicator

## SQLDialect (TReplTargetDatabase)

```
public
  property SQLDialect: Integer read;
DATABASES.SQLDIALECT
```

## SynchronizeTable (TReplTargetDatabase)

```
public
  procedure SynchronizeTable(
      const aSyncRec: TSyncRecord (p.98);
      aSrcDB: TIBDatabase;
      var aCounters: TSyncRecordCounters (p.96) ); virtual; abstract;
```

Performs table synchronization from source database according `aSyncRec` parameters

**See also**

`TIBReplicator` (p.62).`Synchronize` (p.75)

## ReplicateRecord (TReplTargetDatabase)

```
public
  procedure ReplicateRecord(
      const aReplLogRec: TReplLogRecord (p.55);
      aReplOptions: Word;
      aConflict: TStream;
      var aCounter: Integer ); virtual; abstract;
```

Performs replication of aReplLogRec record according `aReplOptions` parameters

`aReplOptions`:

| | |
|---|---|
| See also: | `repoptReplicateLog` (p.103) `repoptReportToSource` (p.103) `repoptReportToTarget` (p.104) `repoptExtConflictCheck` (p.104) `repoptTargetReplication` (p.104) |

`aConflict`:

Returns conflict description if happened

## ObjPrefix (TReplTargetDatabase)

```
public
  ObjPrefix: string;
```

Prefix used for replication objects in database, `DATABASES.OBJPREFIX`

## DBId (TReplTargetDatabase)

```
public
  DBId: Integer;
```

database identifier, `DATABASES.DBID`

## DBMask (TReplTargetDatabase)

```
public
  DBMask: Integer;
```

mask identifying databases in `REPL$LOG`, `SCHEMADB.DBMASK`

## TReplTargetDatabase_IB

```
type
  TReplTargetDatabase_IB = class( TReplTargetDatabase (p.57) );
```
Implements target Interbase database functionality

### Destroy (TReplTargetDatabase_IB)

```
public
  destructor Destroy; override;
```

### SynchronizeTable (TReplTargetDatabase_IB)

```
public
  procedure SynchronizeTable(
      const aSyncRec: TSyncRecord (p.98);
      aSrcDB: TIBDatabase;
      var aCounters: TSyncRecordCounters (p.96) ); override;
```
Performs table synchronization from source database according `aSyncRec` parameters

**See also**

TIBReplicator (p.62).Synchronize (p.75)

### ReplicateRecord (TReplTargetDatabase_IB)

```
public
  procedure ReplicateRecord(
      const aReplLogRec: TReplLogRecord (p.55);
      aReplOptions: Word;
      aConflict: TStream;
      var aCounter: Integer ); override;
```
Performs replication of aReplLogRec record according `aReplOptions` parameters

### Database (TReplTargetDatabase_IB)

```
public
  Database: TIBDatabase;
```

### SnapshotQ (TReplTargetDatabase_IB)

```
public
  SnapshotQ: TIBSQL;
```

## TReplTargetDatabase_Logger

```
type
  TReplTargetDatabase_Logger = class( TReplTargetDatabase (p.57) );
```
Implements target database as text file/logger functionality

### Destroy (TReplTargetDatabase_Logger)

```
public
  destructor Destroy; override;
```

### SynchronizeTable (TReplTargetDatabase_Logger)

```
public
  procedure SynchronizeTable(
      const aSyncRec: TSyncRecord (p.98);
      aSrcDB: TIBDatabase;
      var aCounters: TSyncRecordCounters (p.96) ); override;
```

Performs table synchronization from source database according `aSyncRec` parameters

**See also**

`TIBReplicator` (p.62).`Synchronize` (p.75)

### ReplicateRecord (TReplTargetDatabase_Logger)

```
public
  procedure ReplicateRecord(
      const aReplLogRec: TReplLogRecord (p.55);
      aReplOptions: Word;
      aConflict: TStream;
      var aCounter: Integer ); override;
```

Performs replication of aReplLogRec record according `aReplOptions` parameters

### Logger (TReplTargetDatabase_Logger)

```
public
  Logger: TFileLogger;
```

## TReplLogField_IBSQL

```
type
  TReplLogField_IBSQL = class( TReplLogField (p.54) );
```

Implements source field taken from Interbase

### IsNull (TReplLogField_IBSQL)

```
public
  function IsNull{}: Boolean; override;
```

## TReplLogField_Offline

```
type
  TReplLogField_Offline = class( TReplLogField (p.54) );
```

Implements source field taken from offline package

### IsNull (TReplLogField_Offline)

```
public
  function IsNull{}: Boolean; override;
```

# TReplLogRecord_Online

```
type
  TReplLogRecord_Online = class( TReplLogRecord (p.55) );
```

Implements source log record taken from Interbase

### Create (TReplLogRecord_Online)

```
public
  constructor Create( aReplicator: TIBReplicator (p.62) );
```

### AssignReplQ (TReplLogRecord_Online)

```
public
  procedure AssignReplQ( aReplQ: TIBSQL );
```

### AssignFieldQ (TReplLogRecord_Online)

```
public
  procedure AssignFieldQ( aFieldQ: TIBSQL );
```

### AssignSQ (TReplLogRecord_Online)

```
public
  procedure AssignSQ( SQ: TIBSQL );
```

### Destroy (TReplLogRecord_Online)

```
public
  destructor Destroy; override;
```

### FieldQ_Locate (TReplLogRecord_Online)

```
public
  function FieldQ_Locate(
      aNew: Char;
      aIdx: Integer ): TReplLogField (p.54); override;
```

### FieldQ_IsEmpty (TReplLogRecord_Online)

```
public
  function FieldQ_IsEmpty{}: Boolean; override;
```

# TReplLogRecord_Offline

```
type
  TReplLogRecord_Offline = class( TReplLogRecord (p.55) );
```

Implements source field taken from offline package

### Create (TReplLogRecord_Offline)

```
public
  constructor Create( aReplicator: TIBReplicator (p.62) );
```

### Assign (TReplLogRecord_Offline)

```
public
  procedure Assign(
      const aHeader: TOffHeader (p.90);
      const aReplLog: TOffSrcReplLog (p.91) );
```

### Destroy (TReplLogRecord_Offline)

```
public
  destructor Destroy; override;
```

### FieldQ_Locate (TReplLogRecord_Offline)

```
public
  function FieldQ_Locate(
      aNew: Char;
      aIdx: Integer ): TReplLogField (p.54); override;
```

### FieldQ_IsEmpty (TReplLogRecord_Offline)

```
public
  function FieldQ_IsEmpty{}: Boolean; override;
```

## TIBReplicator

```
type
  TIBReplicator = class( TComponent );
```

Main VCL class that implements IB Replicator functionality. Multi-threading is supported but one TIBReplicator instance can serve only one thread.

### ConfigDatabase (TIBReplicator)

```
published
  property ConfigDatabase: TIBDatabase read write;
```

Configuration database. Set property values before using any method using configuration database. `TIBReplicator` (p.62) uses own `ConfigDatabaseTransaction` (p.64) to access `ConfigDatabase`. Note that for multi-threading is necessary create for each thread separate config database instance and not to use local server `TIBDatabase.DatabaseName` convention. It's limitation of Interbase Express library.

```
  DatabaseName:= 'c:\data\cfgdatabase.gdb'
```

replace with

```
  DatabaseName:= 'localhost:/c:/data/cfgdatabase.gdb'
```

**Example:**

```
procedure TIBReplicationServerForm.FormCreate(Sender: TObject);
begin
...
  IBReplicator.ConfigDatabase.BeforeConnect:= ConfigDatabaseBeforeConnect;
...
end;

procedure TIBReplicationServerForm.ConfigDatabaseBeforeConnect(Sender: TObject);
begin
  with Sender as TIBDatabase do
  begin
    DatabaseName:= IBReplicator.ParseStr(Ini.ReadString(DBIniSection, 'DatabaseName', ''))
    Params.Values['user_name']:= IBReplicator.ParseStr(Ini.ReadString(DBIniSection, 'User'
    Params.Values['password']:= IBReplicator.ParseStr(IBReplicator.ScramblePassword(IBRepl
    SQLDialect:= Ini.ReadInteger(DBIniSection, 'SQLDialect', 3);
  end;
end;
```

## ConfigDatabasePrefix (TIBReplicator)

```
published
  property ConfigDatabasePrefix: string read write;
```

Prefix of objects in configuration database. It's not necessary unless you need use the same database for more replicators or share source/target database with configuration database.

## DBLog (TIBReplicator)

```
published
  property DBLog: TLogger read write;
```

Log used for logging of replication errors and process messages

## ReplLog (TIBReplicator)

```
published
  property ReplLog: TLogger read write;
```

Log used for logging of replication statistics

## LogName (TIBReplicator)

```
published
  property LogName: string read write;
```

Log name used in `TLogger.Log`

## Environment (TIBReplicator)

```
published
  property Environment: TStrings read write;
```

Local replicator environment used by `ParseStr` (p.75) to substitute *$(VAR)* macros

**See also**
`ReadEnvironment` (p.74) `ParseStr` (p.75)

## NowAsUTC (TIBReplicator)

```
published
  property NowAsUTC: Boolean read write;
```

If `True` all timestamps will use UTC. Recommended for real time (non unique time due to daylight saving time) or distributed applications (time zone problem).

## LogErrSQLCmds (TIBReplicator)

```
published
  property LogErrSQLCmds: Boolean read write
      default False;
```

If `True` to `DBLog` (p.63) are logged full SQL commands (raising exception). Useful when adding condition replication trigger commands to obtain where is error.

**See also**
`DBLog` (p.63) `LogErrSQLParams` (p.64)

## LogErrSQLParams (TIBReplicator)

```
published
  property LogErrSQLParams: Boolean read write
      default False;
```

If `True` to `DBLog` (p.63) are logged full query/stored proc parameter values.

**See also**
`DBLog` (p.63) `LogErrSQLCmds` (p.64)

## LogVerbose (TIBReplicator)

```
published
  property LogVerbose: Boolean read write
      default False;
```

If `True` more text is logged.

## TraceSQL (TIBReplicator)

```
published
  property TraceSQL: Boolean read write;
```

Set `True` to enable monitoring of SQL commands using a SQL Monitor

**See also**
`TIBDatabase.TraceFlags`

## ConfigDatabaseTransaction (TIBReplicator)

```
public
  property ConfigDatabaseTransaction: TIBTransaction read;
```

Transaction used for `ConfigDatabase` (p.64)

## CreateConfigDatabase (TIBReplicator)

```
public
  procedure CreateConfigDatabase( aAllowCreate: Boolean );
```

Creates new configuration database or if database file already exists insert/add objects. Note that TIBReplicator first try to open `ConfigDatabase` (p.64) so you can set up `TIBDatabase.Params` in `TIBDatabase.BeforeConnect` event.

`aAllowCreate`:

If database does not exist create new one. Supported only for local databases because uses `FileExists` function.

## UpgradeConfigDatabase (TIBReplicator)

```
public
  procedure UpgradeConfigDatabase;
```

Upgrade `ConfigDatabase` (p.64) that was created by a former version of IBReplicator. The upgrade process is safe, no data should be lost. Note that upgraded database won't be compatible to older IBReplicator.

## CheckConfigDatabaseVersion (TIBReplicator)

```
public
  function CheckConfigDatabaseVersion{}: TDatabaseVersion (p.89);
```

Check if `ConfigDatabase` (p.64) is up to date or is upgradeable.

## UpgradeDatabase (TIBReplicator)

```
public
  procedure UpgradeDatabase( aDBId: Integer );
```

Upgrade source/target database that was used by older version of IBReplicator. The upgrade process is safe, no data should be lost. But it's recommended replicate all data before upgrade process is started. Note that upgraded database won't be compatible to older IBReplicator.

## GenerateFields (TIBReplicator)

```
public
  procedure GenerateFields(
      aSchemaId: Integer;
      aTargetGroupId: Integer );
```

Reads source database and insert relation and fields to the `ConfigDatabase` (p.64). If source database changes (new object was added) it updates information in configuration database. Using is safe - no records will be deleted.

`aSchemaId`:

Schema of which source database will be read

`aTargetGroupId`:

Identifies that will be inserted to `RELATIONS/FIELDS.GROUPID` fields. Target database is not connected. Obsolete if schema defines record history logging.

## CreateSystemObjects (TIBReplicator)

```
public
  procedure CreateSystemObjects(
      aSchemaId: Integer;
      aKeyLength: Integer;
      aGroupId: Integer;
      aDBId: Integer = 0 );
```

### DropSystemObjects (TIBReplicator)

```
public
  procedure DropSystemObjects(
      aSchemaId: Integer;
      aGroupId: Integer;
      aDBId: Integer = 0;
      aOnlyTriggers: Boolean = False );
```

### CreateServerObjects (TIBReplicator)

```
public
  procedure CreateServerObjects(
      aSchemaId: Integer;
      aGroupId: Integer;
      aDBId: Integer = 0 );
```

### DropServerObjects (TIBReplicator)

```
public
  procedure DropServerObjects(
      aSchemaId: Integer;
      aGroupId: Integer;
      aDBId: Integer = 0 );
```

### CreateSystemObjects (TIBReplicator)

```
public
  procedure CreateSystemObjects(
      aDBId: Integer;
      aKeyLength: Integer );
```

Creates / updates all system (replication) objects in database. It prepares database for each schema in which the database participates.

`aDBId`:

database identifier

`aKeyLength`:

Max. length of concatenated primary keys logged into `REPL$LOG`.

**See also**

`DropSystemObjects` (p.66) `CreateServerObjects` (p.67) `DropServerObjects` (p.67) `DeleteSourceSystemData` (p.68)

### DropSystemObjects (TIBReplicator)

```
public
  procedure DropSystemObjects(
      aDBId: Integer;
      aOnlyTriggers: Boolean = False );
```

Removes system objects of all schemes from database

`aDBId`:

database identifier

`aOnlyTriggers`:

- `True` .. remove only replication triggers, helpful when replication condition is changed
- `False` .. all replication objects are dropped and replication records are lost.

**See also**
`CreateSystemObjects` (p.66) `CreateServerObjects` (p.67) `DropServerObjects` (p.67) `DeleteSourceSystemData` (p.68)

## CreateServerObjects (TIBReplicator)

```
public
  procedure CreateServerObjects( aDBId: Integer );
```

Creates server objects - SQL users defined in schemes as replication user.

`aDBId`:
  database identifier

**See also**
`DropServerObjects` (p.67) `CreateSystemObjects` (p.66) `DropSystemObjects` (p.66)

## DropServerObjects (TIBReplicator)

```
public
  procedure DropServerObjects( aDBId: Integer );
```

Removes server objects - SQL users defined in all schemes as replication user.

`aDBId`:
  database identifier

**See also**
`CreateServerObjects` (p.67) `CreateSystemObjects` (p.66) `DropSystemObjects` (p.66)

## CreateStoredProcedureTemplates (TIBReplicator)

```
public
  procedure CreateStoredProcedureTemplates(
      aSchemaId: Integer;
      aGroupId: Integer;
      aDBId: Integer );
```

Creates skeletons of target database stored procedures, i.e. where target relation is selected as *stored procedure*.

`aSchemaId`:
  schema identifier

`aGroupId`:
  group identifier

`aDBId`:
  target database identifier, if `=0` all target databases

## ClearSchemaStatistics (TIBReplicator)

```
public
  procedure ClearSchemaStatistics( aSchemaId: Integer );
```

Clear schema statistics in `ConfigDatabase` (p.64)

**See also**
`ClearRelationStatistics` (p.68)

## ClearRelationStatistics (TIBReplicator)

```
public
  procedure ClearRelationStatistics(
      aSchemaId: Integer;
      aGroupId: Integer;
      aRelationId: Integer );
```

Clear relation statistics in `ConfigDatabase` (p.64)

**See also**
`SchemaRelationStatistics`

## DeleteSourceSystemData (TIBReplicator)

```
public
  procedure DeleteSourceSystemData(
      aSchemaId: Integer;
      aGroupId: Integer;
      aTgtDBId: Integer = 0 );
```

Delete all replication record that belong to selected schema. `REPL$LOG`, `REPL$MAN`, `REPL$TRANSFER`, `REPL$FIELD` are affected. Note that offline packages must be deleted manually.

`aSchemaId`:
  schema identifier

`aGroupId`:
- `=0` .. source database
- `<>0` .. target database

`aTgtDBId`:
  if `aGroupId<>0` .. identifier of target database, if `=0` all target databases

## GetSourceDBId (TIBReplicator)

```
public
  function GetSourceDBId( aSchemaId: Integer ): Integer;
```

Get database id of the source database

## GetSchemaType (TIBReplicator)

```
public
  function GetSchemaType( aSchemaId: Integer ): Integer;
```

Returns type of schema, `SCHEMATA.SCHEMATYPE`

`result`:
See also: `schtReplication` (p.103) `schtRecordHistory` (p.103)

## GetDatabaseName (TIBReplicator)

```
public
  function GetDatabaseName( aDBId: Integer ): string;
```

Returns name of database, `DATABASES.NAME`

## GetDatabaseType (TIBReplicator)

```
public
  function GetDatabaseType( aDBId: Integer ): Integer;
```

Returns type of database, `DATABASES.DBTYPE` `result`:

See also: `dbtInterbase` (p.100) `dbtLog` (p.100)

## GetDBMask (TIBReplicator)

```
public
  function GetDBMask(
      aSchemaId: Integer;
      aGroupId: Integer;
      aDBId: Integer ): Integer;
```

Returns mask that identifies database in `REPL$LOG`, `SCHEMADB.DBMASK`

## AssignDBParams (TIBReplicator)

```
public
  function AssignDBParams(
      aDB: TIBDatabase;
      aId: Integer;
      var aName: string;
      var aObjPrefix: string ): Boolean;
```

Assign database parameters according to values stored in `ConfigDatabase` (p.64) Primary used when opening database for administration.

`aName`:

Database name, `DATABASES.NAME`

`aObjPrefix`:

Prefix of replication objects, `DATABASES.OBJPREFIX`

**See also**

`SetDBParams` (p.70) `SetDBParams_Repl` (p.70)

## SetDBParams (TIBReplicator)

```
public
  procedure SetDBParams(
      aDB: TIBDatabase;
      aId: Integer;
      var aObjPrefix: string );
```

Assign database parameters according to values stored in `ConfigDatabase` (p.64) (`DATABASES`) and connect `aDB` Primary used when opening database for administration.

`aObjPrefix`:

Prefix of replication objects, `DATABASES.OBJPREFIX`

**See also**

`AssignDBParams` (p.69) `SetDBParams_Repl` (p.70)

## SetDBParams_Repl (TIBReplicator)

```
public
  procedure SetDBParams_Repl(
      aDB: TIBDatabase;
      aSchemaId: Integer;
      aGroupId: Integer;
      aDBId: Integer;
      aAlternateTgt: Boolean;
      var aObjPrefix: string );
```

Assign database parameters according to values stored in `ConfigDatabase` (p.64) (`SCHEMADB`) and connect `aDB`. Primary used when opening database for replication.

`aAlternateTgt`:

- `True` .. used when offline database replicates to log file (`dbtLog` (p.100))

`aObjPrefix`:

Prefix of replication objects

**See also**

`AssignDBParams` (p.69) `SetDBParams` (p.70)

## SetDBServiceParams (TIBReplicator)

```
public
  procedure SetDBServiceParams(
      aDBServ: TIBControlService;
      aDBId: Integer;
      var aDatabaseFileName: string );
```

## SetDBServiceParams (TIBReplicator)

```
public
  procedure SetDBServiceParams(
      aDBServ: TIBControlService;
      aDB: TIBDatabase;
      const aDatabaseName: string;
      var aDatabaseFileName: string );
```

Assign Interbase service parameters according to values stored in `ConfigDatabase` (p.64).

`aDatabaseName`:

`DATABASES.NAME`

`aDatabaseFileName`:

Database file name compatible to service, server identifier is stripped

## GetDBProtocol (TIBReplicator)

```
public
  class function GetDBProtocol(
      aDatabaseName: string;
      var aServerName: string;
      var aDatabaseFileName: string ): IBServices.TProtocol;
```

Strip database name

`aDatabaseName`:

Parsed database name

`aServerName`:

Server name, *Local server* if `aDatabaseName` defines a local server database

`aDatabaseFileName`:
: Database file name

`result`:
: Type of protocol

**See also**
TIBDatabase.DatabaseName

## DBSQLExec (TIBReplicator)

```
public
  function DBSQLExec(
      aDB: TIBDatabase;
      aSQL: string;
      aTrans: TIBTransaction = nil ): Integer;
```

Safe SQL execution.

`aTrans`:
: If `nil` .. DefaultTransaction is used. If not in transaction start it.

**See also**
`LogErrSQLCmds` (p.64) `DBLog` (p.63) `DBSQLRecord` (p.71) `SafeExecQuery` (p.73) `SafeExecProc` (p.73)

## DBSQLRecord (TIBReplicator)

```
public
  function DBSQLRecord(
      aDB: TIBDatabase;
      aSQL: string;
      aTrans: TIBTransaction = nil ): Variant;
```

Safe SQL execution.

`aTrans`:
: If `nil` .. aDB.DefaultTransaction is used. If not in transaction start it.

**See also**
`LogErrSQLCmds` (p.64) `DBLog` (p.63) `DBSQLRecord` `SafeExecQuery` (p.73) `SafeExecProc` (p.73)

## SafeStartTransaction (TIBReplicator)

```
public
  procedure SafeStartTransaction( T: TIBTransaction );
```

**See also**
`TIBTransaction.StartTransaction`

## SafeCommit (TIBReplicator)

```
public
  procedure SafeCommit(
      T: TIBTransaction;
      aRetaining: Boolean = False );
```

**See also**
`TIBTransaction.Commit` `TIBTransaction.CommitRetaining`

## SafeRollback (TIBReplicator)

```
public
  procedure SafeRollback(
      T: TIBTransaction;
      aRetaining: Boolean = False;
      aDoNotRaise: Boolean = True );
```

**See also**
`TIBTransaction.Rollback` `TIBTransaction.RollbackRetaining`

## SafeNext (TIBReplicator)

```
public
  procedure SafeNext( Q: TIBSQL );
```

## SafeNext (TIBReplicator)

```
public
  procedure SafeNext( Q: TIBDataSet );
```

**See also**
`TIBDataSet.Next`

## SafeOpen (TIBReplicator)

```
public
  procedure SafeOpen( Q: TIBDataSet );
```

**See also**
`TIBDataSet.Open`

## SafeClose (TIBReplicator)

```
public
  procedure SafeClose(
      Q: TIBDataSet;
      aDoNotRaise: Boolean = True );
```

## SafeClose (TIBReplicator)

```
public
  procedure SafeClose(
      Q: TIBSQL;
      aDoNotRaise: Boolean = True );
```

**See also**
`TIBSQL.Close`

## SafeExecQuery (TIBReplicator)

```
public
  procedure SafeExecQuery( Q: TIBSQL );
```

**See also**
`TIBSQL.ExecQuery`

## SafeExecProc (TIBReplicator)

```
public
  procedure SafeExecProc( Q: TIBStoredProc );
```

**See also**
`TIBStoredProc.ExecProc`

## Now2 (TIBReplicator)

```
public
  function Now2{}: TDateTime;
```

Returns timestamp according to `NowAsUTC` (p.64)

## ClearCachedList (TIBReplicator)

```
public
  procedure ClearCachedList;
```

Clear all cached values read from `ConfigDatabase` (p.64). Values are kept in memory to improve replication performance.

**See also**
`ReadCachedFields` `CachedList`

## DBEnvironmentWriteValue (TIBReplicator)

```
public
  procedure DBEnvironmentWriteValue(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      const aName: string;
      const aValue: string );
```

Writes variable to `REPL$ENVIRONMENT` table

`aName`:
    Variable name

`aValue`:
    Variable value

**See also**
`DBEnvironmentWriteValues` (p.74) `DBEnvironmentReadValue` (p.74) `DBEnvironmentReadValues` (p.74)

## DBEnvironmentReadValue (TIBReplicator)

```
public
  function DBEnvironmentReadValue(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      const aName: string;
      const aDefault: string = '' ): string;
```

Reads variable from `REPL$ENVIRONMENT` table

`aName`:
    Variable name

`aDefault`:
    Value returned if variable not found

**See also**
DBEnvironmentWriteValue (p.74) DBEnvironmentWriteValues (p.74) DBEnvironmentReadValues (p.74)

## DBEnvironmentWriteValues (TIBReplicator)

```
public
  procedure DBEnvironmentWriteValues(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      aValues: TStrings );
```

Writes variables to `REPL$ENVIRONMENT` table

**See also**
DBEnvironmentReadValues (p.74) DBEnvironmentReadValue (p.74) DBEnvironmentWriteValue (p.74)

## DBEnvironmentReadValues (TIBReplicator)

```
public
  procedure DBEnvironmentReadValues(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      aValues: TStrings;
      aSystemKeys: Boolean = False );
```

Reads variables from `REPL$ENVIRONMENT` table

`aSystemKeys`:
If `True` that keys prefixed by an underscore '_' are read.

**See also**
DBEnvironmentWriteValues (p.74) DBEnvironmentWriteValue (p.74) DBEnvironmentReadValue (p.74)

## ReadEnvironment (TIBReplicator)

```
public
  function ReadEnvironment(
      const aName: string;
      const aDefault: string = '';
      aDB: TIBDatabase = nil;
      const aObjPrefix: string = '' ): string;
```

Reads variable from IBReplicator environment if following sequence

- implicit variables (*_VER_*,*_DIR_*,*DBID*,...)
- `Environment` (p.63)
- `REPL$ENVIRONMENT` table in `aDB` database (if not `nil`)
- `ENVIRONMENT` table in `ConfigDatabase` (p.64)
- system environment, keys prefixed *IBREPL_*
- system environment

Implicit variables vary when function is called

- *_DIR_* .. program directory
- *_VER_* .. IBReplicator version
- *SCHEMAID* .. schema id
- *GROUPID* .. groupid
- *DBID* .. database id
- *DB2ID* .. target database in offline replication
- *OFFDIR* .. offline temporary directory

In `ENVIRONMENT` table exists

- *_VERSION* .. database version

`aDefault`:
    returns if variable not found

**See also**
ParseStr

## ParseStr (TIBReplicator)

```
public
  function ParseStr(
      const aPar: string;
      aDB: TIBDatabase = nil;
      const aObjPrefix: string = '' ): string;
```

Searches macros commands in `aPar` and replaces them with environment values. Macro is *$(NAME)* string.

**Example:**

```
ParseStr('$(TEMP)\file.tmp, nil, '')
```

returns *'C:\WINDOWS\TEMP\file.tmp'* because *TEMP* is defined in system environment

```
ParseStr('$(_DIR_)\Log\Repl_$(SCHEMAID)_$(GROUPID)_$(DBID).txt, nil, '')
```

returns e.g. *C:\Program files\IBReplicator\Log\Repl_1_1_2.txt*

## Synchronize (TIBReplicator)

```
public
  procedure Synchronize(
      aSchemaId: Integer;
      const aGroupIds: TIntegerOpenArray (p.89);
      const aTgtDBIds: TIntegerOpenArray (p.89) );
```

Executes source and target database synchronization

`aSchemaId`:
    schema identifier

`aGroupIds`:
    group identifiers (empty to process all groups)

`aTgtDBIds`:
    target database identifiers (empty to process all databases)

## DecodeSyncActions (TIBReplicator)

```
public
  class function DecodeSyncActions(
      const aActS: string;
      aResolve: Boolean ): TSyncActions (p.97);
```

Decodes synchronization actions, `RELATION/SYNCACTION`

## EncodeSyncActions (TIBReplicator)

```
public
  class function EncodeSyncActions( const aActions: TSyncActions (p.97) ): string;
```

Encodes synchronization actions, `RELATION/SYNCACTION`

## ReplicateOnline (TIBReplicator)

```
public
  procedure ReplicateOnline(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      const aSchemaIds: TIntegerOpenArray (p.89);
      const aGroupIds: TIntegerOpenArray (p.89);
      const aTgtDBIds: TIntegerOpenArray (p.89);
      aReplOptions: Word );
```

Executes online replication from whole schema

`aDB`:
  source database

`aObjPrefix`:
  prefix of replication objects in source database

`aSchemaIds`:
  schema identifier(s)

`aGroupIds`:
  group identifiers (empty to process all groups)

`aTgtDBIds`:
  target database identifiers (empty to process all databases)

`aReplOptions`:
  see `TReplTargetDatabase` (p.57).`ReplicateRecord` (p.58)

## SourceOfflineBatch (TIBReplicator)

```
public
  procedure SourceOfflineBatch(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      const aSchemaIds: TIntegerOpenArray (p.89);
      const aGroupIds: TIntegerOpenArray (p.89);
      const aTgtDBIds: TIntegerOpenArray (p.89);
      const aOfflineDir: string;
      aReceiveData: Boolean;
      aProcessReceived: Boolean;
      aProcess: Boolean;
      aSendData: Boolean;
      aResendData: Boolean );
```

Executes selected offline replication actions on source database

`aDB`:
  source database

`aObjPrefix`:
  prefix of replication objects in source database

`aSchemaIds`:
  schema identifier(s)

`aGroupIds`:
  group identifiers (empty to process all groups)

`aTgtDBIds`:
  target database identifiers (empty to process all databases)

`aOfflineDir`:
  offline directory where are stored offline packages. Value is processed using `ParseStr` (p.75)

`aReceiveData`:
  receive acknowledge packages

`aProcessReceived`:
: processed received packages

`aProcess`:
: process records in `REPL$LOG`

`aSendData`:
: send prepared (and pending) offline packages

`aResendData`:
: resend unacknowledged packages

**See also**
`TargetOfflineBatch` (p.78) `ReceivePackages` (p.80) `SendPendingPackages` (p.80) `ProcessReceivedPackages` (p.81)

## PrepareSourceOfflinePackageToFile (TIBReplicator)

```
public
  function PrepareSourceOfflinePackageToFile(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      aSchemaId: Integer;
      aGroupId: Integer;
      aSrcDbId: Integer;
      aTgtDBId: Integer;
      const aOfflineDir: string;
      aIfReplRecord: Boolean ): Integer;
```

process records in `REPL$LOG` and creates offline package

`aDB`:
: source database

`aObjPrefix`:
: prefix of replication objects in source database

`aSchemaId`:
: schema identifier

`aGroupId`:
: schema identifier

`aSrcDbId`:
: source database id

`aTgtDBId`:
: target database id

`aOfflineDir`:
: offline directory where are stored offline packages

`aIfReplRecord`:
- `True` .. generates package only if some record exist in `REPL$LOG`
- `False` .. generates even if no `REPL$LOG` record. Enables sending of 3rd phase acknowledges if log is empty

`result`:
: id of created transfer package

**See also**
`SourceOfflineBatch` (p.76)

## TargetOfflineBatch (TIBReplicator)

```
public
  procedure TargetOfflineBatch(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      const aSchemaIds: TIntegerOpenArray (p.89);
      const aGroupIds: TIntegerOpenArray (p.89);
      const aTgtDBIds: TIntegerOpenArray (p.89);
      const aOfflineDir: string;
      aReceiveData: Boolean;
      aProcess: Boolean;
      aSendData: Boolean;
      aResendData: Boolean;
      aReplOptions: Word );
```

Executes offline replication actions on target database

`aDB`:
    target database

`aObjPrefix`:
    prefix of replication objects in target database

`aSchemaIds`:
    schema identifier(s)

`aGroupIds`:
    group id identifier

`aTgtDBIds`:
    target database identifier (empty to process all databases)

`aOfflineDir`:
    offline directory where are stored offline packages. Value is processed using `ParseStr` (p.75)

`aReceiveData`:
    receive source packages

`aProcess`:
    process received packages

`aSendData`:
    send acknowledge packages

`aResendData`:
    resend unacknowledged packages

`aReplOptions`:
    see `TReplTargetDatabase` (p.57).`ReplicateRecord` (p.58)

**See also**
`SourceOfflineBatch` (p.76) `ProcessSourceOfflinePackageAndPrepareTargetOfflinePackageToFile` (p.79) `ReceivePackages` (p.80) `SendPendingPackages` (p.80) `ProcessReceivedPackages` (p.81)

## ProcessSourceOfflinePackageAndPrepareTargetOfflinePackageToFile (TIBReplicator)

```
public
  function ProcessSourceOfflinePackageAndPrepareTargetOfflinePackageToFile(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      aSchemaId: Integer;
      aGroupId: Integer;
      aSrcDBId: Integer;
      aTgtDBId: Integer;
      const aOfflineDir: string;
      Src: TStream;
      aStampRec: TDateTime;
      aReplOptions: Word;
      var aStamp: TDateTime ): Integer;
```

process offline package and prepare acknowledge package (in target database)

`aDB`:
  target database

`aObjPrefix`:
  prefix of replication object in target database

`aSchemaId`:
  schema identifier

`aGroupId`:
  group identifier

`aSrcDBId`:
  source database identifier

`aTgtDBId`:
  target database identifier

`aOfflineDir`:
  offline directory where are stored offline packages

`Src`:
  source package stream

`aStampRec`:
  timestamp when received

`aReplOptions`:
  see `TReplTargetDatabase` (p.57).`ReplicateRecord` (p.58)

`aStamp`:
  returns timestamp when processed

`result`:
  id of created acknowledge transfer package

**See also**
`TargetOfflineBatch` (p.78)

## DecodePackage (TIBReplicator)

```
public
  procedure DecodePackage(
      Src: TStream;
      var St: TStream );
```

Decodes received package using encoder DLL libraries

## ReceivePackages (TIBReplicator)

```
public
  procedure ReceivePackages(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      aSchemaId: Integer;
      aStatus: Char;
      aGroupId: Integer;
      aSrcDbId: Integer;
      aTgtDBId: Integer;
      const aOfflineDir: string );
```

receive offline packages and store them in offline dir. Receiving is done using transfer library.

`aDB`:
  source/target database

`aObjPrefix`:
  prefix of replication objects

`aSchemaId`:
  schema identifier

`aStatus`:
  type of offline packages (*'S'*,*'T'*)

`aGroupId`:
  group identifier

`aSrcDbId`:
  source database id

`aTgtDBId`:
  target database id

`aOfflineDir`:
  offline directory where are stored offline packages

**See also**
`DecodePackage` (p.79) `SourceOfflineBatch` (p.76) `TargetOfflineBatch` (p.78)

## SendPendingPackages (TIBReplicator)

```
public
  procedure SendPendingPackages(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      aSchemaId: Integer;
      aStatus: Char;
      aGroupId: Integer;
      aSrcDbId: Integer;
      aTgtDBId: Integer;
      const aOfflineDir: string;
      aResendData: Boolean );
```

Receives pending (unsent) offline packages stored in offline dir. Sending is done using encoder and transfer libraries.

`aDB`:
  source/target database

`aObjPrefix`:
  prefix of replication objects

`aSchemaId`:
  schema identifier

`aStatus`:
  type of offline packages (*'S'*,*'T'*)
`aGroupId`:
  group identifier
`aSrcDbId`:
  source database id
`aTgtDBId`:
  target database id
`aOfflineDir`:
  offline directory where are stored offline packages
`aResendData`:
  resend unacknowledged packages

**See also**
`SourceOfflineBatch` (p.76) `TargetOfflineBatch` (p.78)

## ProcessReceivedPackages (TIBReplicator)

```
public
  procedure ProcessReceivedPackages(
      aDB: TIBDatabase;
      const aObjPrefix: string;
      aSchemaId: Integer;
      aStatus: Char;
      aGroupId: Integer;
      aSrcDbId: Integer;
      aTgtDBId: Integer;
      const aOfflineDir: string;
      aReplOptions: Word );
```

Processes received (unprocessed) packages.

`aDB`:
  source/target database
`aObjPrefix`:
  prefix of replication objects
`aSchemaId`:
  schema identifier
`aStatus`:
  type of offline packages (*'S'*,*'T'*)
`aGroupId`:
  group identifier
`aSrcDbId`:
  source database id
`aTgtDBId`:
  target database id
`aOfflineDir`:
  offline directory where are stored offline packages
`aReplOptions`:
  see `TReplTargetDatabase` (p.57).`ReplicateRecord` (p.58)

**See also**
`SourceOfflineBatch` (p.76) `TargetOfflineBatch` (p.78)

## ExecSQLScript (TIBReplicator)

```
public
  procedure ExecSQLScript(
      aSchemaId: Integer;
      aGroupId: Integer;
      aDBId: Integer;
      aFromSeqId: Integer;
      aToSeqId: Integer;
      aSnapshot: Boolean;
      aFromDT: TDateTime;
      aToDT: TDateTime;
      aSQL: TStream;
      var aCntTotal: Integer;
      var aCntProc: Integer;
      var aCntBlob: Integer );
```

Procedure processes a SQL script to a source/target database. If the SQL script is a log made by replicator that contains special tags (SEQID, stamp and BLOB content) condition may by applied and the BLOB fields are updated. Table `REPL$SNAPSHOT` is also affected. If no condition is used all commands are processed.

`aSchemaId`:
: schema identifier

`aGroupId`:
: group identifier

`aDBId`:
- `= 0` .. process to source database
- `> 0` .. process to target database

`aFromSeqId`:
: low bound of SEQID condition, zero to minus infinity

`aToSeqId`:
: high bound of SEQID condition, zero to infinity

`aSnapshot`:
: if `True` and SEQID is found in SQL log then update `REPL$SNAPSHOT` table (if SEQID is greater than id found in `REPL$SNAPSHOT` table).

`aFromDT`:
: low bound of datetime condition, zero to minus infinity

`aToDT`:
: high bound of datetime condition, zero to infinity

`aSQL`:
: a stream containing the SQL script

`aCntTotal`:
: number of SQL commands found in SQL script

`aCntProc`:
: number of processed SQL commands (according condition and success)

`aCntBlob`:
: number of processed BLOBs meta SQL commands

## BackupDatabase (TIBReplicator)

```
public
  procedure BackupDatabase(
      aDB: TIBDatabase;
      const aDatabaseName: string;
      const aBackupFileName: string;
      aOptions: TBackupOptions = [ ConvertExtTables ] );
```

## RestoreDatabase (TIBReplicator)

```
public
  procedure RestoreDatabase(
      aDB: TIBDatabase;
      const aDatabaseName: string;
      const aBackupFileName: string;
      aOptions: TRestoreOptions = [ Replace ];
      aPageSize: Integer = 4096 );
```

## BackupDatabase (TIBReplicator)

```
public
  procedure BackupDatabase(
      aDBId: Integer;
      const aBackupFileName: string;
      aOptions: TBackupOptions = [ ConvertExtTables ] );
```

Backups database to `GBAK` compatible format.

`aDBId`:
  Database identifier

`aBackupFileName`:
  Backups database to `GBAK` compatible format.

See also: `RestoreDatabase` (p.83)

`aOptions`:
  Backup options, see IBX documentation

**See also**
`RestoreDatabase` (p.83)

## RestoreDatabase (TIBReplicator)

```
public
  procedure RestoreDatabase(
      aDBId: Integer;
      const aBackupFileName: string;
      aOptions: TRestoreOptions = [ Replace ];
      aPageSize: Integer = 4096 );
```

Restores database from `GBAK` compatible format. If database already exists data in database is replaced and data lost.

`aDBId`:
  Database identifier

`aBackupFileName`:
  Restore input file name. Value is processed using `ParseStr` (p.75)

`aOptions`:
  Restore options, see IBX documentation

`aPageSize`:
Database page size

**See also**
`BackupDatabase` (p.83)

### CloneSourceDatabase (TIBReplicator)

```
public
  procedure CloneSourceDatabase(
      aSchemaId: Integer;
      aGroupId: Integer;
      aDBId: Integer;
      const aMetadataOnly: Boolean );
```
Procedure clones source database, removes all replication objects, replaces target database and creates replication objects. If target database already exists is replaced and data are lost.

`aSchemaId`:
schema identifier

`aGroupId`:
group identifier

`aDBId`:
target database id

`aMetadataOnly`:
if `True` no data will be added, cloned database remains empty

### ScramblePassword (TIBReplicator)

```
public
  class function ScramblePassword(
      aPass: string;
      aEncode: Boolean ): string;
```
Scrambles password not to be human readable

`aEncode`:
scramble/unscramble switch

### IsScrambled (TIBReplicator)

```
public
  class function IsScrambled( const aPass: string ): Boolean;
```
Returns `True` if `aPass` is already scrambled

### OfflinePackageToXML (TIBReplicator)

```
public
  function OfflinePackageToXML(
      aSrc: TStream;
      aHeaderOptions: Byte ): UTF8String;
```
Converts content of an offline package to XML file according *offpackage.dtd*.

`aSrc`:
offline package

`aHeaderOptions`:
- `0` .. do not write *<?xml version ...?>* header - `1` .. do not write *<!DOCTYPE ...* > header - `2` .. write headers

**See also**
ConflictToXML (p.85)

## ConflictToXML (TIBReplicator)

```
public
  function ConflictToXML(
      Src: TStream;
      aHeaderOptions: Byte ): UTF8String;
```

Converts content of an conflict description to XML file according *conflict.dtd*. Description is stored in `REPL$MAN.CONFLICT`.

`aHeaderOptions`:

- `0` .. do not write *<?xml version ...?>* header - `1` .. do not write *<!DOCTYPE ... >* header - `2` .. write headers

**See also**
OfflinePackageToXML (p.84)

## Create (TIBReplicator)

```
public
  constructor Create( aOwner: TComponent ); override;
```

## Destroy (TIBReplicator)

```
public
  destructor Destroy; override;
```

## OnUpdateStatus (TIBReplicator)

```
published
  property OnUpdateStatus: TOnUpdateStatusEvent (p.96) read write;
```

Event called from `ReplicateOnline` (p.76), `Synchronize` (p.75) and `ProcessSourceOfflinePackageAndPrepareTargetOfflinePackage` to inform application about progress.

## ManualStop (TIBReplicator)

```
public
  ManualStop: Boolean;
```

Enabled to stop a long term replication/synchronization process. Note that is periodically processed message queue. Set true to stop processing.

**See also**
TApplication.ProcessMessages

# EIBReplicatorError

```
type
  EIBReplicatorError = class( Exception );
```

Replicator exception class

**See also**
IBReplicatorError (p.88)

### Kind (EIBReplicatorError)

```
published
  Kind: Byte;
```

- 1 .. replication conflict
- 2 .. replication error
- 0 .. other

## IBDT2DT

```
function IBDT2DT( S: string ): TDateTime;
```

Converts an Interbase datetime to TDateTime, international settings independent

## DT2IBDT

```
function DT2IBDT( DT: TDateTime ): string;
```

Converts TDateTime to Interbase datetime, international settings independent

## DT2OffStamp

```
function DT2OffStamp( DT: TDateTime ): TOffStamp (p.89);
```

Converts TDateTime to TOffStamp

## OffStamp2DT

```
function OffStamp2DT( S: TOffStamp (p.89) ): TDateTime;
```

Converts TOffStamp to TDateTime

## NowUTC

```
function NowUTC{}: TDateTime;
```

Returns datetime in UTC (universal time coordinate) (=GMT .. Greenwich mean time)

## Blob2SafeSQL

```
function Blob2SafeSQL( const S: string ): string;
```

Strip dangerous chars from blob (used for logging to SQL text script)

## CheckIBConnectionError

```
function CheckIBConnectionError(
    E: Exception;
    aDB: TIBDatabase ): Boolean;
```

Check if exception is raised due a DB connection problem.

`E`:
 Exception instance from `except` clause

`aDB`:
 database to force close if connection is lost

See also: `TIBDatabase.ForceClose`

`result`:
 `True` if connection is lost

## CheckDatabaseName

```
procedure CheckDatabaseName( const aDatabaseName: string );
```

Checks if `aDatabaseName` is not *Local server* database when current thread is not `MainThread`.

**See also**

Server settings (p.29) Database properties (p.17)

## IntegerToIntegerArray

```
function IntegerToIntegerArray( const I: Integer ): TIntegerOpenArray (p.89);
```

## NullIntegerArray

```
function NullIntegerArray{}: TIntegerOpenArray (p.89);
```

## StringArrayToIntegerArray

```
function StringArrayToIntegerArray( const aArr: TStringOpenArray (p.89) ): TIntegerOpenArray
```

Converts a TStringOpenArray to TIntegerOpenArray

## IntegerArrayToStringArray

```
function IntegerArrayToStringArray( const aArr: TIntegerOpenArray (p.89) ): TStringOpenArray
```

Converts a TIntegerOpenArray to TStringOpenArray

## StringToArray

```
function StringToArray(
    S: string;
    aSeparator: Char ): TStringOpenArray (p.89);
```

Converts string to TStringOpenArray

`aSeparator`:
 Identifies character that separates items in string

## ArrayToString

```
function ArrayToString(
    const aArr: TStringOpenArray (p.89);
    const aSep: string ): string;
```

Converts a TStringOpenArray to string

`aSep`:

Identifies character that will separate items in string

## IntegerArrayToSQLCondition

```
function IntegerArrayToSQLCondition(
    const aArr: TIntegerOpenArray (p.89);
    const aField: string;
    const aPrefix: string = '';
    const aSuffix: string = '' ): string;
```

Creates SQL condition according `aArr`. If array is empty returns empty string, if aArr is one-item array returns *FIELD=x*, if there are more items returns *FIELD IN (x,y,z)*.

`aPrefix`:

Prefix added prior to result

`aSuffix`:

Prefix added to result

## IBReplicatorError

```
procedure IBReplicatorError(
    const Msg: string;
    aKind: Byte );
```

Raises `EIBReplicatorError` (p.85)

`Msg`:

Message

`aKind`:

- 1..replication conflict
- 2..replication error
- 0..other error (don't write to `REPL$MAN`)

## Register

```
procedure Register;
```

Use by IDE to register components

## TDatabaseVersion

```
type
  TDatabaseVersion =
    ( verCurrent
    , verUpgradable
    , verNonUpgradable
    );
```

Used by `TIBReplicator` (p.62).`CheckConfigDatabaseVersion` (p.65) to detect config database compatibility

`verCurrent`:
  Config databases is up to date

`verUpgradable`:
  Config database is not up to date but is upgradable using `TIBReplicator` (p.62).`UpgradeConfigDatabase` (p.65)

`verNonUpgradable`:
  Config database cannot be upgraded automatically

## TStringOpenArray

```
type
  TStringOpenArray = array of string;
```

An open array that stores list of strings

## TIntegerOpenArray

```
type
  TIntegerOpenArray = array of Integer;
```

An open array that stores list of integers

## TOffStamp

```
type
  TOffStamp = array[ 1 .. 9 ] of Char;
```

Platform independent timestamp (no double, float cross platform incompatibility)

## TRelName

```
type
  TRelName = array[ 1 .. 30 ] of Char;
```

A name of relation of a Interbase object

## TOffHeader

```
type
  TOffHeader = record
    Magic: TOffMagic;
    SchemaId: LongInt;
    TransferId: LongInt;
    Status: Char;
    SrcDBId: LongInt;
    TgtDBId: LongInt;
    GroupId: LongInt;
    Stamp: TOffStamp (p.89);
    Crc: TCRC32;
    Reserved: array[ 0 .. 15 + 7 ] of Byte;
  end {TOffHeader};
```

A header of an offline package

`Magic`:
  Magic number identifying offline package

`SchemaId`:
  Schema contained in offline package

`TransferId`:
  Transfer identifier

`Status`:
- `'S'` .. package created on the source side
- `'T'` .. package created on the target side

`SrcDBId`:
  Source database identifier

`TgtDBId`:
  Target database identifier

`GroupId`:
  Database group identifier

`Stamp`:
  Timestamp when package was created

`Crc`:
  CRC of the package

`Reserved`:
  Reserved for future use

## TOffSrcRelDefField

```
type
  TOffSrcRelDefField = record
    FieldType: Byte;
    Name: TRelName (p.89);
  end {TOffSrcRelDefField};
```

Field definition in the source relation

`FieldType`:
  Type of field
- 1..primary field
- 2..foreign field
- 3..normal field

`Name`:

Field name

**See also**

`TOffSrcRelDef` (p.91)

## TOffSrcRelDef

```
type
  TOffSrcRelDef = record
    RecType: Byte;
    RelationId: LongInt;
    Name: TRelName (p.89);
    Count: Word;
  end {TOffSrcRelDef};
```

Source relation definition

`RecType`:

Must be equal to `offrtRelationDef` (p.102)

`RelationId`:

Relation identifier

`Name`:

Relation name

`Count`:

Count of fields in relation
Follows:

```
Fields: array[0..Count-1] of TOffSrcRelDefField;
```

**See also**

`TOffSrcRelDefField` (p.90)

## TOffSrcReplLog

```
type
  TOffSrcReplLog = record
    RecType: Byte;
    RelationId: LongInt;
    SeqId: LongWord;
    RepType: Char;
    Sep: Char;
    Stamp: TOffStamp (p.89);
    Options: Byte;
  end {TOffSrcReplLog};
```

Structure contains one `REPL$LOG` record.
First occurrence must be proceeded by a relation definition (p.91)

`RecType`:

Must be equal to `offrtReplLog` (p.102)

`RelationId`:

Relation identifier

`SeqId`:
  Seq. identifier from `REPL$LOG.SEQID`
`RepType`:
  Replication type from `REPL$LOG.REPLTYPE`
`Sep`:
  Separator used as delimiter of primary keys. Equivalent of `REPL$LOG.SEPARATOR`
`Stamp`:
  Timestamp from `REPL$LOG.STAMP`
`Options`:
  See `optxxxx` constants
  Follows:

```
Keys: array of Char;  // string(OLDKEY, NEWKEY, NEWFKEY (in new package is empty))
if Options and optIsForeignRecord <> 0
begin
  NewForeignRecord: TOffSrcRelRec;
  OldForeignRecord: TOffSrcRelRec;
end;
if Options and optIsRelRecord <> 0
  RelRecord: TOffSrcRelRec;
```

  If `IsRelRecord <> 0` then follows:
See also: optIsRelRecord optIsForeignRecord optUTF8

**See also**
TOffSrcRelRec (p.92)

## TOffSrcRelRec

```
type
  TOffSrcRelRec = record
    FieldCount: Word;
  end {TOffSrcRelRec};
```

Structure contains one source relation record
`FieldCount`:
  Count of fields in array that follows:

```
Values: array[0..Count-1] of TOffReplRecValue;
```

**See also**
TOffReplRecValue (p.92)

## TOffReplRecValue

```
type
  TOffReplRecValue = record
    DataType: Char;
    FieldType: Byte;
  end {TOffReplRecValue};
```

Value of a field in source relation

`DataType`:

Type of a value that follows:

- `N` .. null
- `C` .. string(65535), follows `Len: Word; Wide/UTF8 String`
- `M` .. memo longstring, follows `Len: LongWord; Wide/UTF8 String`
- `O` .. blob/binary, follows `Len: LongWord; Data`
- `D` .. date, follows `yyyymmdd`
- `T` .. time, follows `hhnnss`
- `S` .. datetime, follows `yyyymmddhhnnsszzz`
- `B` .. Byte, follows `Byte`
- `I` .. Integer, follows `LongInt`
- `J` .. Int64, follows `Int64`
- `F` .. float, follows `extended`
- `' '` .. field not found, follows `Len: longword; widestring`

`FieldType`:

Type of field `TFieldType`

## TOffTgtReplAckRec

```
type
  TOffTgtReplAckRec = record
    Flag: Byte;
    SeqId: LongInt;
  end {TOffTgtReplAckRec};
```

Result of record replication

`Flag`:

See `ackflxxx` constants

See also: ackflManual ackflReasonStr ackflConflict

Follows:

```
if Flags and ackflReasonStr <> 0 then
  Len: Word; ReasonStr: UTF8String;
if Flags and ackflConflict <> 0 then
begin
  Conflict: TOffTgtReplAckConflictRec;
end;
```

`SeqId`:

Sequential id of corresponding `REPL$LOG.SEQID` record

## TOffTgtReplAckConflictRec

```
type
  TOffTgtReplAckConflictRec = record
    Flag: Byte;
    Stamp: TOffStamp (p.89);
    TgtDBId: LongInt;
    Count: Word;
  end {TOffTgtReplAckConflictRec};
```

The structure describes record conflict

`Flag`:

Reason of conflict

See also: confoptPrimaryKeyViolation confoptRecordViolation confoptFieldViolation

`Stamp`:
: Stamp when conflict occurred

`TgtDBId`:
: Target database identifier

`Count`:
: Number of fields if record
: Follows:

```
TgtRelationName: string }
Values: array[0..Count-1] of packed record
begin
  SrcFieldName: string[Length(SrcFieldName)];
  SrcValue: TOffReplRecValue;
  TgtFieldName: string[Length(TgtFieldName)];
  TgtValue: TOffReplRecValue;
end;
```

**See also**
`TOffTgtReplAckRec` (p.93)

## TOffTgtReplAck

```
type
  TOffTgtReplAck = record
    RecType: Byte;
    TransferIdAck: LongInt;
    StampRec: TOffStamp (p.89);
    StampProc: TOffStamp (p.89);
    ProcCount: LongInt;
  end {TOffTgtReplAck};
```

Record contained in 2nd phase acknowledge package

`RecType`:
: Must be equal to `offrtReplAck` (p.102)

`TransferIdAck`:
: Identified of the source transfer id

`StampRec`:
: Timestamp when source package was received

`StampProc`:
: Timestamp when source package was processed

`ProcCount`:
: Number of processed records
: Follows:

```
SeqIds: array[0..ProcCount-1] of TOffTgtReplAckRec;
```

**See also**
`TOffTgtReplAckRec` (p.93)

## TOffSrcReplAckAck

```
type
  TOffSrcReplAckAck = record
    RecType: Byte;
    TransferId: LongInt;
    TransferIdAck: LongInt;
    StampSent: TOffStamp (p.89);
    StampRec: TOffStamp (p.89);
    StampProc: TOffStamp (p.89);
  end {TOffSrcReplAckAck};
```

Record contained in 3rd phase acknowledge package

`RecType`:
  Must be equal to `offrrtReplAckAck` (p.102)

`TransferId`:
  Identified of 1st pass transfer package

`TransferIdAck`:
  Identifier of 2nd phase transfer package

`StampSent`:
  Timestamp when 1st phase package was sent

`StampRec`:
  Timestamp when 2nd phase acknowledge was received

`StampProc`:
  Timestamp when 2nd phase was processed

## TOffTgtReplAckAckAck

```
type
  TOffTgtReplAckAckAck = record
    RecType: Byte;
    TransferIdAck: LongInt;
    StampSent: TOffStamp (p.89);
  end {TOffTgtReplAckAckAck};
```

Record contained in 4th phase acknowledge package

`RecType`:
  Must be equal to `offrrtReplAckAckAck` (p.102)

`TransferIdAck`:
  Identifier of 2nd phase transfer package

`StampSent`:
  Timestamp when 2nd phase package was sent

## TOffReplResend

```
type
  TOffReplResend = record
    RecType: Byte;
    TransferId: LongInt;
    TransferIdAck: LongInt;
    StampRec: TOffStamp (p.89);
    StampProc: TOffStamp (p.89);
  end {TOffReplResend};
```

Record used for `REPL$TRANSFER` record

`RecType`:
  Must be equal to `offrReplResend` (p.103)

`TransferId`:
  `REPL$TRANSFER.TRANSFERID`, identifier of record

`TransferIdAck`:
  `REPL$TRANSFER.TRANSFERID_ACK`

`StampRec`:
  `REPL$TRANSFER.STAMP_REC`

`StampProc`:
  `REPL$TRANSFER.STAMP_PROC`

## TOnUpdateStatusEvent

```
type
  TOnUpdateStatusEvent = procedure (
      Sender: TObject;
      aTicks: LongWord;
      aCnt: LongWord ) of object;
```

Prototype of a function called when `TIBReplicator` (p.62) is working

`aTicks`:
  Number of ticks elapsed since start

`aCnt`:
  Number of processed records

**Example:**

```
// indicate replication performance at status bar
 procedure TIBReplDataModule.UpdateStatusLine(Sender: TObject; aTicks, aCnt: Longword);
 begin
   StatusBar.Panels[1].Text:= IntToStr(aTicks div 1000);
   StatusBar.Panels[2].Text:= IntToStr(aCnt);
   if aTicks = 0 then
     StatusBar.Panels[3].Text:= '0'
   else
     StatusBar.Panels[3].Text:= FloatToStrF(aCnt / (aTicks / 1000 / 60), ffGeneral, 2, 0);
 end;
```

## TSyncRecordCounters

```
type
  TSyncRecordCounters = record
    Total: Integer;
    Inserted: Integer;
    Updated: Integer;
    Deleted: Integer;
    Errors: Integer;
    Conflicts: Integer;
    StartTick: LongWord;
  end {TSyncRecordCounters};
```

Structure to store synchronization counters

`Total`:
  Total number of processed records
`Inserted`:
  Number of inserted records
`Updated`:
  Number of updated records
`Deleted`:
  Number of deleted records
`Errors`:
  Number of errors
`Conflicts`:
  Number of conflicts
`StartTick`:
  Tick when synchronization started

**See also**

`TReplTargetDatabase` (p.57).`SynchronizeTable` (p.58)

## TSyncAction

```
type
  TSyncAction = record
    Action: Byte;
    Where_Src: string;
    Where_Tgt: string;
  end {TSyncAction};
```

Structure identifying synchronization action

`Action`:

See also: `saUpd` (p.99) `saInsDel` (p.100) `saSrcTgt` (p.100) `saRO` (p.100)

**See also**

`TReplTargetDatabase` (p.57).`SynchronizeTable` (p.58)

## TSyncCondition

```
type
  TSyncCondition = record
    Alias: string;
    Where: string;
  end {TSyncCondition};
```

## TSyncActions

```
type
  TSyncActions = record
    Acts: array of TSyncAction (p.97);
    Cond: array of TSyncCondition (p.97);
  end {TSyncActions};
```

Open array list of actions

## TSyncRecord

```
type
  TSyncRecord = record
    SchemaId: Integer;
    GroupId: Integer;
    SrcDBId: Integer;
    TgtDBId: Integer;
    RelationId: Integer;
    TgtDBName: string;
    RelationName: string;
    Tag: Integer;
    Action: TSyncAction (p.97);
  end {TSyncRecord};
```

Structure passing config database parameters to `TReplTargetDatabase` (p.57).`SynchronizeTable` (p.58)

`Tag`:

  Internal use only

`Action`:

  Action description

## TReplFieldItem

```
type
  TReplFieldItem = record
    FieldName_Src: string;
    FieldName_Tgt: string;
    FieldId: Integer;
    Updatable_Tgt: Boolean;
    Options: Integer;
    Idx: Integer;
    Tag: Integer;
  end {TReplFieldItem};
```

Structure caching config database parameters

`Tag`:

  Internal use only

**See also**
`TReplFieldDefs` (p.52)

## TReplFieldRef

```
type
  TReplFieldRef = record
    FieldType: Byte;
    Idx: Integer;
  end {TReplFieldRef};
```

**See also**
`TReplFieldDefs` (p.52)

## PReplOffRecordBuffer

```
type
  PReplOffRecordBuffer = ^TReplOffRecordBuffer;
```

## TReplOffRecordBuffer

```
type
  TReplOffRecordBuffer = record
    Buffer: TMemoryStream;
    UTF8: Boolean;
    Pos: array of Integer;
  end {TReplOffRecordBuffer};
```

Internal structure to store offline record values

**See also**
TReplLogRecord_Offline (p.62)

## TIBObjectType

```
type
  TIBObjectType =
    ( ibobjRelation
    , ibobjProcedure
    , ibobjTrigger
    , ibobjFunction
    , ibobjGenerator
    , ibobjRole
    );
```

Used in TIBReplicator (p.62).CreateObject, TIBReplicator (p.62).DropObject and TIBReplicator (p.62).ObjectExists functions

`ibobjRelation`:
Identifies a relation (table)

`ibobjProcedure`:
Identifies a stored procedure

`ibobjTrigger`:
Identifies a trigger

`ibobjFunction`:
Identifies an external function

`ibobjGenerator`:
Identifies a generator

`ibobjRole`:
Identifies a role

## saUpd

```
const
  saUpd = '01';
```

Bit mask, update during synchronization enabled

**See also**
`TSyncAction` (p.97)

## saInsDel

```
const
  saInsDel = '02';
```

Bit mask, insert/delete during synchronization enabled

**See also**
`TSyncAction` (p.97)

## saSrcTgt

```
const
  saSrcTgt = '04';
```

Bit mask, loop target relation
if bit = `0` then loop in source database and find target record (can insert new records) if bit = `0` then loop in target database and find source record (can delete non-existing records)

**See also**
`TSyncAction` (p.97)

## saRO

```
const
  saRO = '08';
```

Bit mask, don't update target database but create differential log

**See also**
`TSyncAction` (p.97)

## dbtInterbase

```
const
  dbtInterbase = 0;
```

equivalent of `DATABASES.DBTYPE`, instructs that database is Interbase

**See also**
`TIBReplicator` (p.62).`GetDatabaseType` (p.68)

## dbtLog

```
const
  dbtLog = 1;
```

equivalent of `DATABASES.DBTYPE`, instructs that database text file

**See also**

`TIBReplicator` (p.62).`GetDatabaseType` (p.68)

## optIsRelRecord

```
const
  optIsRelRecord = '01';
```

Bit mask, instructs that `TOffSrcReplLog` (p.91) contains relation record

## optIsForeignRecord

```
const
  optIsForeignRecord = '02';
```

Bit mask, instructs that `TOffSrcReplLog` (p.91) contains foreign record

## optUTF8

```
const
  optUTF8 = '04';
```

Bit mask, instructs that `TOffSrcReplLog` (p.91) strings are encoded as UTF8, alternatively are encoded in Unicode

## ackflManual

```
const
  ackflManual = '01';
```

Bit mask of TOffTgtReplAckRec.Flag

- `0` .. delete record from `REPL$LOG`
- `1` .. move record to `REPL$MAN`

**See also**

TOffTgtReplAckRec

## ackflReasonStr

```
const
  ackflReasonStr = '02';
```

Bit mask applied to `TOffTgtReplAckRec` (p.93).Flag, instructs that structure contains reason of replication error

## ackflConflict

```
const
  ackflConflict = '04';
```

Bit mask applied to `TOffTgtReplAckRec` (p.93).Flag, instructs that structure contains conflict description

## offrtRelationDef

```
const
  offrtRelationDef = 1;
```

**See also**
`TOffSrcRelDef` (p.91)

## offrtReplLog

```
const
  offrtReplLog = 2;
```

**See also**
`TOffSrcReplLog` (p.91)

## offrtReplAck

```
const
  offrtReplAck = 3;
```

**See also**
`TOffTgtReplAck` (p.94)

## offrrtReplAckAck

```
const
  offrrtReplAckAck = 4;
```

**See also**
`TOffSrcReplAckAck` (p.95)

## offrrtReplAckAckAck

```
const
  offrrtReplAckAckAck = 5;
```

**See also**
`TOffSrcReplAckAck` (p.95)

## offrReplResend

```
const
  offrReplResend = 6;
```

**See also**
TOffReplResend (p.95)

## schtReplication

```
const
  schtReplication = 0;
```

equivalent of `SCHEMA.DBTYPE`, instructs that schema defines a replication

**See also**
TIBReplicator (p.62).GetSchemaType (p.68)

## schtRecordHistory

```
const
  schtRecordHistory = 1;
```

equivalent of `SCHEMA.DBTYPE`, instructs that schema defines a record history logging

**See also**
TIBReplicator (p.62).GetSchemaType (p.68)

## repoptReplicateLog

```
const
  repoptReplicateLog = '01';
```

Bit mask, do not replicate records but move `REPL$LOG` from source to target database

**See also**
TReplTargetDatabase (p.57).ReplicateRecord (p.58)

## repoptReportToSource

```
const
  repoptReportToSource = '02';
```

Bit mask, report problem to source database `REPL$MAN`

**See also**
TReplTargetDatabase (p.57).ReplicateRecord (p.58)

## repoptReportToTarget

```
const
  repoptReportToTarget = '04';
```

Bit mask, report problems to target `REPL$MAN`

**See also**

`TReplTargetDatabase` (p.57).`ReplicateRecord` (p.58)

## repoptExtConflictCheck

```
const
  repoptExtConflictCheck = '08';
```

Bit mask, enable extended conflict checking

**See also**

`TReplTargetDatabase` (p.57).`ReplicateRecord` (p.58)

## repoptTargetReplication

```
const
  repoptTargetReplication = '10';
```

Bit mask, replicate from target database `REPL$LOG` to target database

**See also**

`TReplTargetDatabase` (p.57).`ReplicateRecord` (p.58)

## confoptPrimaryKeyViolation

```
const
  confoptPrimaryKeyViolation = 1;
```

Replicator cannon find equivalent record in target database when updating or deleting, or target record already exists when inserting

**See also**

`TOffTgtReplAckConflictRec` (p.93) `repoptExtConflictCheck` (p.104) `fldoptLogConflict` (p.106)

## confoptRecordViolation

```
const
  confoptRecordViolation = 2;
```

A field conflict of fields marked `fldoptDoNotUpdateRecordWhenConflict` (p.106) preventing record modification

**See also**

`TOffTgtReplAckConflictRec` (p.93) `repoptExtConflictCheck` (p.104) `fldoptTargetPriorityConflict` (p.105) `fldoptLogConflict` (p.106) `fldoptDoNotUpdateRecordWhenConflict` (p.106)

## confoptFieldViolation

```
const
  confoptFieldViolation = 3;
```

A field has not been updated due field conflict. Non conflicting fields in record are updated.

**See also**
`TOffTgtReplAckConflictRec` (p.93) `repoptExtConflictCheck` (p.104) `fldoptTargetPriorityConflict` (p.105) `fldoptLogConflict` (p.106) `fldoptDoNotUpdateRecordWhenConflict` (p.106)

## reloptWipeLogOnDelete

```
const
  reloptWipeLogOnDelete = '01';
```

Bit mask, instructs replicator to delete are non processed `REPL$LOG` records when record is deleted in source database. It prevents replication of records that are inserted/modified and then deleted before changes are replicated to targets. Equivalent of `RELATIONS.OPTIONS`.

## fldoptDoNotInsert

```
const
  fldoptDoNotInsert = '01';
```

Bit mask, instructs never insert such fields when inserting.
Equivalent of `FIELDS.OPTIONS`.

## fldoptDoNotUpdate

```
const
  fldoptDoNotUpdate = '02';
```

Bit mask, instructs never update such fields when updating.
Equivalent of `FIELDS.OPTIONS`.

## fldoptUpdateOnlyWhenChanged

```
const
  fldoptUpdateOnlyWhenChanged = '04';
```

Bit mask, instructs allow update such fields only when was changed in source database. Note that `repoptExtConflictCheck` (p.104) must be enabled.
Equivalent of `FIELDS.OPTIONS`.

## fldoptTargetPriorityConflict

```
const
  fldoptTargetPriorityConflict = '08';
```

Bit mask, instructs leave field value unchanged when field conflict occurs. Note that `repoptExtConflictCheck` (p.104) must be enabled.

Equivalent of `FIELDS.OPTIONS`.

## fldoptLogConflict

```
const
  fldoptLogConflict = '10';
```

Bit mask, instructs log conflict to `REPL$MAN`
Equivalent of `FIELDS.OPTIONS`.

## fldoptDoNotUpdateRecordWhenConflict

```
const
  fldoptDoNotUpdateRecordWhenConflict = '20';
```

Bit mask, prevents record modification if particular field conflict occurs. Note that `repoptExtConflictCheck` (p.104) must be enabled.
Equivalent of `FIELDS.OPTIONS`.

## fldoptDoNotDeleteRecordWhenConflict

```
const
  fldoptDoNotDeleteRecordWhenConflict = '100';
```

Bit mask, prevents record delete if particular field conflict occurs. Note that `repoptExtConflictCheck` (p.104) must be enabled.
Equivalent of `FIELDS.OPTIONS`.

## tranRO

```
const
  tranRO = 'read'#13#10'read_committed';
```

default read only transaction parameters

## tranRW

```
const
  tranRW = 'write'#13#10'read_committed'#13#10'rec_version'#13#10'wait';
```

default read/write transaction parameters