Client Side Journaling (Experimental)

Client side journaling creates a per database attachment file recording each read/write transaction start and end, and SQL query that modifies data in the database. It can optionally record each read only transaction and queries that do not modify data (e.g. select queries).

The purpose of client side journaling is:

1. To create transaction and query log that can be used to recover from a lost database connection or server crash, and
2. To provide a record of database activity for debugging purposes.

Database Schema Dependencies

The following table and generator must be present in the database in order to support journaling:

```
    Create Table IBX$JOURNALS(
      IBX$SessionID Integer not null,
      IBX$TransactionID Integer not null,
      IBX$OldTransactionID Integer,
      IBX$USER VarChar(32) Default CURRENT_USER,
      IBX$TIMESTAMP TIMESTAMP Default CURRENT_TIMESTAMP,
      Primary Key(IBX$SessionID,IBX$TransactionID)
    );
  CREATE SEQUENCE IBX$SESSIONS;
```

If these tables are not present when journaling is enabled, the client will attempt to create the tables. However, it is recommended that they are created as part of the database schema in order to avoid having to give potentially every client table creation privilege.

Using Client Side Journaling

Client side journaling is controlled through the following IAttachment functions:

```
{Journaling options. Default is [joReadWriteTransactions,joModifyQueries] }

   TJournalOption = (joReadOnlyTransactions, joReadWriteTransactions,
                     joModifyQueries, joReadOnlyQueries);
   TJournalOptions = set of TjournalOption;

     function JournalingActive: boolean;
     function GetJournalOptions: TJournalOptions;
    function StartJournaling(aJournalLogFile: AnsiString): integer; overload;
    function StartJournaling(aJournalLogFile: AnsiString;
                         Options: TJournalOptions): integer; overload;
    procedure StopJournaling(RetainJournal: boolean);
```

Call StartJournaling to enable journaling to the specified file.

- The Options determine the scope of the journaling. By default, only read/write transactions and queries that modify the database data are journaled. This can be extended to include read only and other queries by specifying an appropriate set of options.

When the function is called, a unique session id is obtained using the database's IBX$SESSIONS generator. This is returned by the function.

When StopJournaling is called, or the attachment closed, the journal file is closed. If RetainJournal is false then the Journal File is deleted, and the IBX$JOURNALS entry for the current session removed.

Note: StopJournaling is called implicitly when the database connection is closed or the database dropped:

- If the database connected is closed as a result of the interface going out of scope the RetainJournal is true
- On a Forced disconnection, RetainJournal is true.
- If the database connection is closed by DropDatabase then RetainJournal is false.

Journal File Contents

The journal file is a text file and is intended to be human readable. Each record in the journal starts with an asterisk and is terminated by a line separator. Each record is identified by a single character immediately after the asterisk, followed by a timestamp, session id and transaction id. Note that the timestamp uses the current default format settings for shortdate and longtime, extended to four decimal places.

The following records are defined:

Transaction Start:

***S:<date/time>,<session id>,<transaction no.>,<string length>:<transaction Name>,<string length>:<TPB>,<default Completion>**

This journal entry records when ITransaction.StartTransaction is called (explicitly or implicitly). It additionally records:

- The transaction number as returned by Firebird
- A Local Transaction Name as specified when the ITransaction interface is created.
- The Transaction Parameter Block (TPB) in text format, and
- the default completion (Rollback (0) or Commit(1)).

A row in the IBX$JOURNALS table is also created with the session id and transaction no as its primary key. The row is created using the newly started transaction and will only become visible to other transactions once the transaction is committed. The idea is that this entry can be used to determine if a transaction in the journal file has or has not been fully committed to the database.

Transaction Commit ：

***C:<date/time>,<session id>,<transaction no.>**

This journal entry records when ITransaction.Commit is called (explicitly or implicitly).

Transaction Commit retaining：

***c:<date/time>,<session id>,<transaction no.>,<old transaction no.>**

This journal entry records when ITransaction.CommitRetaining is called.

A new IBX$JOURNALS entry is made recording both the new transaction no. and the old transaction no. Note that while the context is retained, a new transaction is otherwise started following a commit retaining, and has a new transaction no.

Transaction Rollback:

***R:<date/time>,<session id>,<transaction no.>**

This journal entry records when ITransaction.Rollback is called (explicitly or implicitly).

Transaction Rollback retaining:

***r:<date/time>,<session id>,<transaction no.>,<old transaction no.>**

This journal entry records when ITransaction.RollbackRetaining is called.

A new IBX$JOURNALS entry is made recording both the new transaction no. and the old transaction no. Note that while the context is retained, a new transaction is otherwise started following a commit retaining, and has a new transaction no.

Update/Insert/Delete

***Q:<date/time>,<session id>,<transaction no.>,<length of query text in bytes>:<query text>**

This journal entry records a query execution. The query text is the original SQL query with the parameter names or placeholders substituted with the literal value of each parameter. In the case of binary blobs and arrays, the TIBXScript syntax is used.

Reading the Journal File

The journal is intended to be both human and machine readable. In order to support programmatic parsing of the journal file, the class TJournalProcessor may be found in the IBUtils unit. The following is an example of use:

```
uses Classes, Sysutils, IBUtils;

TMyClass = class
private
  procedure HandleOnJnlEntry(JnlEntry: TJnlEntry);
  procedure PrintTPB(TPB: ITPB);
public
  procedure PrintJournalFile(aFileName: AnsiString);
end;

procedure TMyClass.PrintJournalFile(aFileName: AnsiString);
begin
 writeln('Journal Entries');
 with TJournalProcessor.Create do
 try
    Execute(aFileName,FirebirdAPI,HandleOnJnlEntry);
 finally
   Free
 end;
end;

procedure TMyClass.HandleOnJnlEntry(JnlEntry: TJnlEntry);
begin
 with JnlEntry do
 begin
   {$IFNDEF FPC}
   writeln('Journal Entry = ',ord(JnlEntryType),'(',
     TJournalProcessor.JnlEntryText(JnlEntryType),')');
   {$ELSE}
     writeln('Journal Entry = ',JnlEntryType,'(',
     TJournalProcessor.JnlEntryText(JnlEntryType),')');
   {$ENDIF}
   writeln('Timestamp = ',
                      FBFormatDateTime('yyyy/mm/dd hh:nn:ss.zzzz',Timestamp));
   writeln('Session ID = ',SessionID);
   writeln('Transaction ID = ',TransactionID);
   case JnlEntry.JnlEntryType of
```

```
    jeTransStart:
      begin
        writeln('Transaction Name = "',TransactionName,'"');
        PrintTPB(TPB);
        {$IFNDEF FPC}
        writeln('Default Completion = ',ord(DefaultCompletion));
        {$ELSE}
        writeln('Default Completion = ',DefaultCompletion);
        {$ENDIF}
      end;

    jeQuery:
      begin
        writeln('Query = ',QueryText);
      end;

    jeTransCommitRet,
    jeTransRollbackRet:
      writeln('Old TransactionID = ',OldTransactionID);
    end;
  end;
  writeln;
end;

procedure TMyClass.PrintTPB(TPB: ITPB);
var i: integer;
begin
  writeln('TPB: Item Count = ', TPB.getCount);
  for i := 0 to TPB.getCount - 1 do
  begin
    write('  ',TPB[i].getParamTypeName);
    if TPB[i].AsString <> '' then
      writeln(' = ', TPB[i].AsString)
    else
      writeln;
  end;
  writeln;
end;
```