Firebird Wire Protocol

Carlos Guzman Alvarez, Mark Rotteveel

Version 0.16, 13 April 2025

Table of Contents

Chapter 1. Introduction

This document describes the Firebird wire protocol. Most of the information was obtained by studying the Firebird source code and implementing the wire protocol in the Firebird .NET provider and Jaybird (Firebird JDBC driver).

The protocol is described in the form of the message sent by the client and received from the server. The described protocol is Firebird/Interbase protocol version 10. Earlier (Interbase) versions of the protocol are not in scope for this document. Changes in later protocol versions are described in notes below the description of the relevant version 10 message (currently higher versions are only partially described).

This document is not complete. It is advisable to consult the *InterBase 6 API Guide* for additional information on subjects like parsing the status vector, information request items, and the meaning of operations. You can find this manual under “InterBase 6.0 Manuals” in the Reference Manuals section of the Firebird website.

Unless otherwise indicated, a client request must be flushed to the server for processing. For some operations the flush can be deferred, so it is sent together with a different operation. Versions 11 and higher of the wire protocol explicitly support (or even require) deferring of operations, including deferring the read of the response.

In the protocol descriptions below, we include the names of the fields of the structs used in the Firebird sources; this can make it easier to search for how it’s used in Firebird itself.

Chapter 2. Responses

The wire protocol has a limited set of responses. Some operations have a specific response, which is described together with the operation. Most operation however use one (or more) of the responses described in this section. The meaning and content depend on the operation that initiated the response.

2.1. Generic response

`Int32` — `p_operation`

Operation code

If operation equals `op_response`:

`Int32` — `p_resp_object`

Object handle

Although 32-bit, valid handle values are always between 0 and 65535 (0xFFFF), with the "normal" range between 0 and 65000, where `0` either represents the connection itself, or means "no value".

`Int64` — `p_resp_blob_id`

Object ID

`Buffer` — `p_resp_data`

Data (meaning depends on the operation).

`Byte[]` — `p_resp_status_vector`

Status vector

The format of the status vector is basically `<tag><value>[{tag><value>} ...]<end>`, with `<tag>` an `Int32`, and where parsing of `<value>` depends on `<tag>`; `<end>` is `Int32` `isc_arg_end` — `0`. The length can only be determined by correctly parsing the status vector. The first 8 bytes are always an `Int32` tag (`isc_arg_gds` or `isc_arg_warning`) and an `Int32` value.

- If the status vector starts with `Int32` `isc_arg_gds` — `1` **and** the second `Int32` is non-zero, it is a failure response.
- If it starts with `Int32` `isc_arg_warning` — `18` **and** the second `Int32` is non-zero, it is a success response with warning(s).
- Otherwise, if the second `Int32` is zero, it is a success response

Information about parsing the status vector can be found in the *Interbase 6 API Guide* in the documentation set. It might also be advantageous to look at the sources of Firebird .NET Data Provider or Jaybird.

2.2. SQL response

Success response to `op_execute2` (see Execute) or `op_executeimmediate2` (not yet documented).

`Int32` — `p_operation`
: Operation code

If operation equals `op_sql_response`:

`Int32` — `p_sqldata_messages`
: Count of rows following response (in practice, only `1` or `0`)

Row data
: The row data is not in a buffer like described in Data types, but as a sequence (0..1) of data rows with a special format, see Reading row data.

  You can also consider the row data not a part of the SQL response, but something that is sent **after** the SQL response.

2.3. Fetch response

Success response to `op_fetch` (see Fetch) and `op_fetch_scroll` (not yet documented).

`Int32` — `p_operation`
: Operation code

If operation equals `op_fetch_response`:

`Int32` — `p_sqldata_status`
: Status

  - `0` — success
  - `100` — end of cursor

`Int32` — `p_sqldata_messages`
: Count of rows following response (in practice, only `1` or `0`)

  A value of `0` indicates end-of-batch (fetch complete). Together with status `100`, it also means end-of-cursor, otherwise there are more rows available for a next fetch.

Row data
: The row data is not in a buffer like described in Data types, but as a sequence (0..1) of data rows with a special format, see Reading row data.

  You can also consider the row data not a part of the fetch response, but something that is sent **after** the fetch response.

The success response to Fetch is not a single of `op_fetch_response`, but a sequence of `op_fetch_response` and row data. That is:

```
<op-fetch-response (status = 0, count = 1)>
<row-data>
<op-fetch-response (status = 0, count = 1)>
<row-data>
...
if end-of-cursor:
  <op-fetch-response (status = 100, count = 0)>
else:
  <op-fetch-response (status = 0, count = 0)>
```

Firebird may return fewer rows than requested in Fetch.

2.4. Slice response

Success response to Get slice.

This documentation might not reflect actual encoding in the protocol.

Response to Get slice.

`Int32` — `p_operation`
: Operation code

If operation equals `op_slice`:

`Int32` — `p_slr_length`
: Slice length

`Int32`
: Slice length (possibly a buffer?, needs verification)

`Buffer`
: Slice data

Chapter 3. Databases

3.1. Attach

Attachments to a database are done in two steps, first identification (connect) to the server, then attach to a database.

3.1.1. Identification

The identification and attach handshake changed significantly in protocol 13 (Firebird 3.0), and is not yet documented.

Performs the initial handshake and protocol selection.

Client

`Int32` — `p_operation`

Operation code (`op_connect`)

`Int32` — `p_cnct_operation`

Operation code; unused in practice, can always be `0`. Some implementations use `op_attach` (19) for historic(?) reasons.

`Int32` — `p_cnct_cversion`

Connect version:

`CONNECT_VERSION2` — `2` user identification encoding is undefined (Firebird 1.0 — Firebird 2.5)

`CONNECT_VERSION3` — `3` user identification is UTF-8 encoded (since Firebird 3.0 and higher, but backwards compatible as the version wasn't checked before Firebird 3.0)

`Int32` — `p_cnct_client`

Architecture type (e.g. `arch_generic` = 1).

`String` — `p_cnct_file`

Database path or alias

The encoding of this is undefined, which can lead to problems with non-ASCII paths if the server and client use a different encoding.

`Int32` — `p_cnct_count`

Count of protocol versions understood (e.g. 1).

`Buffer` — `p_cnct_user_id`

User identification

TODO: Needs further description

The next block of data declares the protocol(s) that the client is willing or able to support. It should be sent as many times as protocols are supported (and specified in `p_cnct_count` above). Values depend on the protocol.

If a client sends more than 10 (Firebird 5.0 and older) or 11 (Firebird 6.0) protocols, the surplus are ignored.

Int32 — p_cnct_version

Protocol version (`PROTOCOL_VERSION10`)

Int32 — p_cnct_architecture

Architecture type (e.g. `arch_generic = 1`)

It is possible to use a different architecture value, but then connection is only possible with a server of the same architecture. In addition, it changes how responses and/or data needs to be parsed or encoded (the authors don't know the exact details). In short, use `arch_generic`.

Int32 — p_cnct_min_type

Minimum type (e.g. `ptype_batch_send = 3`)

Possible values:

ptype_page — 1	Page server protocol (never supported in Firebird)
ptype_rpc — 2	Simple remote procedure call (not supported since Firebird 3.0)
ptype_batch_send — 3	Batch sends, no asynchrony
ptype_out_of_band — 4	Batch sends w/ out of band notification (semantics not documented in this manual)
ptype_lazy_send — 5	Deferred packets delivery

Int32 — p_cnct_max_type

Maximum type (e.g. `ptype_lazy_send — 5`)

If the client wants to set up wire compression, this `ptype`-code must be OR'ed with `pflag_compress` (`0x100`). See also discussion below for server response.

Int32 — p_cnct_weight

Preference weight (e.g. 2). Higher values have higher preference. For equal weights, the last supported occurrence will be selected.

Server

Success response:

`Int32` — `p_operation`

Operation code

If operation equals `op_accept`:

`Int32` — `p_acpt_version`

Protocol version number accepted by server

`Int32` — `p_acpt_architecture`

Architecture for protocol

`Int32` — `p_acpt_type`

Accepted type and additional flags. Obtain the type by masking with `0xFF`.

Known flags:

`pflag_compress` — `0x100`	Turn on compression In the request from client to server, it signals a request to use wire compression. In the response from the server to client, it is an acknowledgement, and wire compression **must** be enabled *after* processing this response.
`pflag_win_sspi_nego` — `0x200`	Win_SSPI supports Negotiate security package Only sent from server to client.

Failure response: Generic response

3.1.2. Attachment

Attaches to a database. Attach is the same as Create (`op_create`), but using `op_attach` instead of `op_create`.

Client

`Int32` — `p_operation`

Operation code (`op_attach`)

`Int32` — `p_atch_database`

Database object id; unused in practice, can always be `0`.

`String` — `p_atch_file`

Database path or alias

If `isc_dpb_utf8_filename` is present in the database parameter buffer below, the encoding is UTF-8; otherwise, the encoding is undefined. The `isc_dpb_utf8_filename` item is supported since Firebird 2.5.

`Buffer` — `p_atch_dpb`

Database parameter buffer

Table 1. Example of parameters sent in the DPB

Parameter	Description	Value	Optional
`isc_dpb_version1`	Version (must be first item!)		
`isc_dpb_dummy_packet_interval`	Dummy packet interval	120	*
`isc_dpb_sql_dialect`	SQL dialect	3	
`isc_dpb_lc_ctype`	Character set	UTF8	
`isc_dpb_sql_role_name`	User role	RDB$ADMIN	*
`isc_dpb_connect_timeout`	Connection timeout	10	*
`isc_dpb_user_name`	User name	SYSDBA	
`isc_dpb_password`	User password	masterkey	

Server

Generic response

3.2. Detach

Detaches from the database. After detach the connection is still open, to disconnect use Disconnect (`op_disconnect`).

Client

`Int32` — `p_operation`

Operation code (`op_detach`)

`Int32` — `p_rlse_object`

Database handle (always `0`)

Server

Generic response

3.3. Create

Create a database. Create is the same as Attachment (`op_attach`), but using `op_create` instead of `op_attach`.

Client

`Int32` — `p_operation`

Operation code (`op_create`)

`Int32` — `p_atch_database`

Database object id; unused in practice, can always be `0`.

`String` — `p_atch_file`

Database path or alias

If `isc_dpb_utf8_filename` is present in the database parameter buffer below, the encoding is UTF-8; otherwise, the encoding is undefined. The `isc_dpb_utf8_filename` item is supported since Firebird 2.5.

There are a number of DPB items to configure the newly created database, including page size (`isc_dpb_page_size`) — which cannot be modified after creation.

`Buffer` — `p_atch_dpb`

Database parameter buffer

Server

Generic response

The `CREATE DATABASE` statement

Although Firebird has a `CREATE DATABASE` statement, the documented syntax is not fully supported by Firebird server. Part of the syntax (e.g. database name, user, password, page size) are parsed by *fbclient* to execute the `op_create` (or equivalent for embedded).

After the database is successfully created, *fbclient* then uses execute immediate (`op_execute_immediate`) without transaction to execute a reduced `CREATE DATABASE` statement for additional configuration of the database.

3.4. Drop

Drops the currently attached database.

Client

`Int32` — `p_operation`

Operation code (`op_drop_database`)

`Int32` — `p_rlse_object`

Database handle

Server

Generic response

3.5. Database information request

Requests database or server information.

Client

`Int32` — `p_operation`

Operation code (`op_info_database`)

`Int32` — `p_info_object`

Database handle; unused in practice, can always be `0`.

`Int32` — `p_info_incarnation`

Incarnation of object (`0`)

TODO: Usage and meaning?

`Buffer` — `p_info_items`

Requested information items

Values of enum `db_info_types` in Firebird's `inf_pub.h`.

`Int32` — `p_info_buffer_length`

Length of buffer available for receiving response

Too small may lead to receiving a truncated buffer, which necessitates requesting information again with a larger size.

The buffer in the response is sized to the actual length of the response (upto the declared available length), so specifying a larger than necessary size does not inflate the response on the wire.

Server

Generic response — on success, `p_resp_data` holds the requested information.

+ A truncated response is considered a success, and can only be determined by parsing `p_resp_data`.

3.6. Disconnect

Client

`Int32` — `p_operation`

Operation code (`op_disconnect`)

No response, remote socket close.

Closing the connection (socket) without sending an `op_disconnect` will result in "Connection reset by peer" (error `10054` (Windows) or `104` (Linux)) in `firebird.log`.

Chapter 4. Transactions

4.1. Start transaction

Starts a transaction with the transaction options specified in the transaction parameter buffer.

Client

`Int32` — `p_operation`

Operation code (`op_transaction`)

`Int32` — `p_sttr_database`

Database handle; unused in practice, can always be `0`.

`Buffer` — `p_sttr_tpb`

Transaction parameter buffer

Server

Generic response — on success, `p_resp_object` is the new transaction handle.

The `SET TRANSACTION` statement

Instead of using `op_transaction` to start a transaction, it is also possible to use the `SET TRANSACTION` statement.

This statement needs to be executed with execute immediate (`op_execute_immediate`) without transaction. On success, the `p_resp_object` holds the transaction handle.

4.2. Commit transaction

Commits an active or prepared transaction.

Client

`Int32` — `p_operation`

Operation code (`op_commit`)

`Int32` — `p_rlse_object`

Transaction handle

Server

Generic response

4.3. Rollback transaction

Rolls back an active or prepared transaction.

Client

`Int32` — **`p_operation`**

Operation code (`op_rollback`)

`Int32` — **`p_rlse_object`**

Transaction handle

Server

Generic response

4.4. Commit retaining

Commits an active or prepared transaction, retaining the transaction context.

Client

`Int32` — **`p_operation`**

Operation code (`op_commit_retaining`)

`Int32` — **`p_rlse_object`**

Transaction handle

Server

Generic response

4.5. Rollback retaining

Rolls back an active or prepared transaction, retaining the transaction context.

Client

`Int32` — **`p_operation`**

Operation code (`op_rollback_retaining`)

`Int32` — **`p_rlse_object`**

Transaction handle

Server

Generic response

4.6. Prepare

Performs the first stage of a two-phase commit. After prepare, a transaction is *in-limbo* until committed or rolled back.

4.6.1. Simple prepare

Client

`Int32` — `p_operation`

Operation code (`op_prepare`)

`Int32` — `p_rlse_object`

Transaction handle

Server

Generic response

4.6.2. Prepare with message

Associates a message (byte data) with the prepared transaction. This information is stored in `RDB$TRANSACTIONS` and can be used for recovery purposes.

Client

`Int32` — `p_operation`

Operation code (`op_prepare2`)

`Int32` — `p_prep_transaction`

Transaction handle

`Buffer` — `p_prep_data`

Recovery information

Server

Generic response

4.7. Transaction information request

This is similar to Database information request.

Client

`Int32` — `p_operation`

Operation code (`op_info_transaction`)

`Int32` — `p_info_object`

Transaction handle

`Int32` — `p_info_incarnation`

Incarnation of object (`0`)

TODO: Usage and meaning?

`Buffer` — `p_info_items`

Requested information items

Values of constants in Firebird's `inf_pub.h` starting with `isc_info_tra_` or `fbinfo_tra_`.

`Int32` — `p_info_buffer_length`

Length of buffer available for receiving response

Too small may lead to receiving a truncated buffer, which necessitates requesting information again with a larger size.

The buffer in the response is sized to the actual length of the response (upto the declared available length), so specifying a larger than necessary size does not inflate the response on the wire.

Server

Generic response — on success, `p_resp_data` holds the requested information.

+ A truncated response is considered a success, and can only be determined by parsing `p_resp_data`.

Chapter 5. Statements

5.1. Allocate

Allocates a statement handle on the server.

Client

`Int32` — `p_operation`

Operation code (`op_allocate_statement`)

`Int32` — `p_rlse_object`

Database handle

Server

Generic response — on success, `p_resp_object` is the allocated statement handle.

5.1.1. Deviations for protocol version 11

An `op_allocate_statement` can only be sent together with a Prepare operation.

This may not be entirely correct, and needs further verification.

5.2. Free

Frees resources held by the statement.

Client

`Int32` — `p_operation`

Operation code (`op_free_statement`)

`Int32` — `p_sqlfree_statement`

Statement handle

`Int32` — `p_sqlfree_option`

Option	Description
`DSQL_close` – 1	Closes the cursor opened after statement execute.
`DSQL_drop` – 2	Releases the statement handle.

Option	Description
`DSQL_unprepare` – `4`	*Firebird 2.5 or higher* Close resources associated with statement handle, and unprepares the currently allocated statement text. The statement handle itself is retained. + It is not necessary to unprepare before preparing a new statement on the same handle.

Server

Generic response

5.2.1. Deviations for protocol version 11

Request flushing and response processing can be deferred for `ptype_batch_send` or higher.

For `DSQL_drop` and `DSQL_unprepare`, we recommend flushing immediately so the server at least processes the request, which will prevent longer than necessary retention of metadata locks. The response processing can then still be deferred until receiving the response of another action.

5.3. Prepare

Client

`Int32` — `p_operation`

Operation code (`op_prepare_statement`)

`Int32` — `p_sqlst_transaction`

Transaction handle

`Int32` — `p_sqlst_statement`

Statement handle

`Int32` — `p_sqlst_SQL_dialect`

SQL dialect (1 or 3)

This should generally match the connection dialect.

`String` — `p_sqlst_SQL_str`

Statement to be prepared

`Buffer` — `p_sqlst_items`

Describe and describe bind information items

Example of requested information items

- `isc_info_sql_select`
- `isc_info_sql_describe_vars`

- `isc_info_sql_sqlda_seq`
- `isc_info_sql_type`
- `isc_info_sql_sub_type`
- `isc_info_sql_length`
- `isc_info_sql_scale`
- `isc_info_sql_field`
- `isc_info_sql_relation`

`Int32` — **`p_sqlst_buffer_length`**

Target buffer length (`32768`)

Server

Generic response — on success, `p_resp_data` holds the statement description (matching the requested information items)

For statements with a lot of columns and/or parameters, it may be necessary to handle truncation of the buffer by repeating the describe and/or describe bind information request using Information request and using `isc_info_sql_sqlda_start` to inform the server from which column or parameter to continue.

For an example, see Jaybird's `StatementInfoProcessor.handleTruncatedInfo(…)`.

5.3.1. Deviations for protocol version 11

The statement handle can no longer be allocated separately. The initial Allocate operation **must** be sent together with the first prepare operation. When allocating and preparing together, the value of the statement handle of the *prepare* must be `0xFFFF` (invalid object handle). The responses must be processed in order: first *allocate* response, then *prepare* response.

Once a statement handle has been allocated, it can be reused by sending a *prepare* with the obtained statement handle.

5.4. Describe

Requesting a description of output parameters (columns) of a query is done using the statement information request message

Example of requested information items

- `isc_info_sql_select`
- `isc_info_sql_describe_vars`
- `isc_info_sql_sqlda_seq`
- `isc_info_sql_type`
- `isc_info_sql_sub_type`

- `isc_info_sql_length`
- `isc_info_sql_scale`
- `isc_info_sql_field`
- `isc_info_sql_relation`

The initial request can be done as part of Prepare. The information can be requested together with Describe bind (input parameters).

5.5. Describe bind (input parameters)

Describe of input parameters of a query is done using the statement information request message

Example of requested information items

- `isc_info_sql_select`
- `isc_info_sql_describe_vars`
- `isc_info_sql_sqlda_seq`
- `isc_info_sql_type`
- `isc_info_sql_sub_type`
- `isc_info_sql_length`
- `isc_info_sql_scale`
- `isc_info_sql_field`
- `isc_info_sql_relation`

The initial request can be done as part of Prepare. The information can be requested together with Describe.

5.6. Execute

Client

`Int32` — `p_operation`

Operation code

Operation	Usage
`op_execute`	DDL and DML statements
`op_execute2`	Executable stored procedures, or singleton `RETURNING` (i.e. statements described as `isc_info_sql_stmt_exec_procedure`)

`Int32` — `p_sqldata_statement`

Statement handle

`Int32` — `p_sqldata_transaction`

Transaction handle

`Buffer` — `p_sqldata_blr`

Parameters in BLR format

If there are no parameters, send a zero-length buffer.

`Int32` — `p_sqldata_message_number`

Message number; unused, always use `0`

`Int32` — `p_sqldata_messages`

Number of messages — 1 if there are parameters, `0` if there are no parameters

`Buffer` — ***no name***

Parameter values

If `p_sqldata_messages` is `0`, this buffer must not be sent (not even as a zero-length buffer)

TODO: Might not even be a buffer, verify.

If using `op_execute2` (the statement is a stored procedure and there are output parameters):

`Buffer` — `p_sqldata_out_blr`

Output parameters in BLR format

`Int32` — `p_sqldata_out_message_number`

Output message number (0) ??

Additions in protocol 16 and higher

`UInt32` — `p_sqldata_timeout`

Statement timeout value in milliseconds (`0` — use connection-level statement timeout)

Additions in protocol 18 and higher

`UInt32` — `p_sqldata_cursor_flags`

Cursor flags

`CURSOR_TYPE_SCROLLABLE` — `0x01` request scrollable cursor

Additions in protocol 19 and higher

`UInt32`-- `p_sqldata_inline_blob_size`

Maximum inline blob size

A value of `0` disables inline blobs. The server may use a lower limit than requested. In the Firebird 5.0.3 and Firebird 6 implementation at the time of writing, the server has a maximum of 65535 bytes.

TODO: Describe `op_inline_blob` somewhere

Server

For `op_execute`:

Generic response

For `op_execute2`:

Success response: SQL response followed by Generic response

Failure response: only Generic response

5.7. Rows affected by query execution

Obtaining the rows affected by a query is done using the statement information request message

List of requested information items

- `isc_info_sql_records`

5.8. Fetch

Client

`Int32` — `p_operation`
: Operation code (`op_fetch`)

`Int32` — `p_sqldata_statement`
: Statement handle

`Buffer` — `p_sqldata_blr`
: Output parameters in BLR format

  Only needs to be sent on first fetch; subsequent fetches can send a zero-length buffer.

`Int32` — `p_sqldata_message_number`
: Message number (always `0`)

`Int32` — `p_sqldata_messages`
: Message count/fetch size (e.g. `200`)

  The server may decide to return fewer rows than requested, even if the end-of-cursor wasn't reached yet.

Server

Success response: one or more Fetch response

Failure response: Generic response — with an error in `p_resp_status_vector`

It is possible to receive Generic response with an error in the status vector after one or more fetch responses.

5.9. Set cursor name

Client

`Int32` — `p_operation`
: Operation code (`op_set_cursor`)

`Int32` — `p_sqlcur_statement`
: Statement handle

`String` — `p_sqlcur_cursor_name`
: Cursor name (null terminated!)

`Int32` — `p_sqlcur_type`
: Cursor type

  Reserved for future use, always use `0`.

Server

Generic response

5.10. Information request

This is similar to Database information request.

Client

`Int32` — `p_operation`
: Operation code (`op_info_sql`)

`Int32` — `p_info_object`
: Statement handle

`Int32` — `p_info_incarnation`
: Incarnation of object (`0`)

  TODO: Usage and meaning?

`Buffer` — `p_info_items`
: Requested information items

  Values of constants in Firebird's `inf_pub.h` starting with `isc_info_sql_`.

`Int32` — `p_info_buffer_length`

Length of buffer available for receiving response

Too small may lead to receiving a truncated buffer, which necessitates requesting information again with a larger size.

The buffer in the response is sized to the actual length of the response (upto the declared available length), so specifying a larger than necessary size does not inflate the response on the wire.

Server

Generic response — on success, `p_resp_data` holds the requested information.

+ A truncated response is considered a success, and can only be determined by parsing `p_resp_data`.

Information about how to parse the information buffer sent by the Firebird server can be found in the InterBase 6.0 documentation set

Chapter 6. Blobs

6.1. Create/Open

Client

`Int32` — `p_operation`

Operation code

Operation	Description
`op_create_blob`	Creates a new blob
`op_create_blob2`	Creates a new blob with a blob parameter buffer
`op_open_blob`	Opens an existing blob
`op_open_blob2`	Opens an existing blob with a blob parameter buffer

`Buffer` — `p_blob_bpb`

Blob parameter buffer

Only sent for `op_create_blob2` and `op_open_blob2`

`Int32` — `p_blob_transaction`

Transaction handle

`Int64` — `p_blob_id`

Blob ID

Server

Generic response — on success

+

a. `p_resp_object` is the blob handle

b. `p_resp_blob_id` is the blob id (for `op_create_blob` / `op_create_blob2` only)

6.1.1. Deviations for protocol version 11

Request flushing and response processing can be deferred.

If `ptype_batch_send` or higher is used, other blob operations can be sent immediately after the open/create. They can use the *invalid object* handle 0xFFFF instead of the — not yet received — blob handle.

6.2. Get segment

Client

`Int32` — `p_operation`

Operation code (`op_get_segment`)

`Int32` — `p_sgmt_blob`

Blob handle

`Int32` — `p_sgmt_length`

Segment length

Maximum length is 32767 for Firebird 2.5 and older, 65535 for Firebird 3.0 and higher.

`Buffer` — `p_sgmt_segment`

Always a zero-length buffer

Server

Generic response — on success, `p_resp_data` is the blob segment

The response buffer in `p_resp_data` contains zero or more segments. Each segment starts with 2-bytes for the length (little-endian), followed by that length of data.

6.2.1. Deviations for protocol version 11

Request flushing and response processing can be deferred.

If `ptype_batch_send` or higher is used, `op_get_segment` can be batched with Create/Open (and other blob operations) by using the *invalid object* handle 0xFFFF.

6.3. Put segment

Client

`Int32` — `p_operation`

Operation code (`op_put_segment`)

`Int32` — `p_sgmt_blob`

Blob handle

`Int32` — `p_sgmt_length`

Length of segment data (effectively ignored; possibly only in recent Firebird versions)

`Buffer` — `p_sgmt_segment`

Blob segment

If the blob was created as a segmented blob, the maximum length is 32765 (Firebird 2.5 and

older) or 65533 (Firebird 3.0 and higher).

For stream blobs, there is no length limitation other than the maximum buffer length (TODO: verify, might only be for recent versions).

Server

Generic response

6.3.1. Deviations for protocol version 11

Request flushing and response processing can be deferred.

If `ptype_batch_send` or higher is used, `op_put_segment` can be batched with Create/Open (and other blob operations) by using the *invalid object* handle 0xFFFF.

6.4. Batch segments

Similar to Put segment, but allows to send multiple segments.

Client

`Int32` — `p_operation`

Operation code (`op_batch_segments`)

`Int32` — `p_sgmt_blob`

Blob handle

`Int32` — `p_sgmt_length`

Length of segment data (effectively ignored; possibly only in recent Firebird versions)

`Buffer` — `p_sgmt_segment`

Blob segments

The buffer can contain one or more segments, which are prefixed with 2 bytes of length (little endian), followed by the data. The maximum length per segment is 32765 (Firebird 2.5 and older) or 65533 (Firebird 3.0 and higher).

Server

Generic response

6.4.1. Deviations for protocol version 11

Request flushing and response processing can be deferred.

If `ptype_batch_send` or higher is used, `op_batch_segment` can be batched with Create/Open (and other blob operations) by using the *invalid object* handle 0xFFFF.

6.5. Seek

Seek is only supported for blobs that were created as a stream blob. Seek is not fully supported for blobs longer than 2 GiB (4 GiB?).

Client

`Int32` — `p_operation`
: Operation code (`op_seek_blob`)

`Int32` — `p_seek_blob`
: Blob handle

`Int32` — `p_seek_mode`
: Seek mode

`blb_seek_from_head` — `0`	absolute seek from start of blob
`blb_seek_relative` — `1`	relative seek from current position
`blb_seek_from_tail` — `2`	absolute seek from end of blob

`Int32` — `p_seek_offset`
: Offset

Server

Generic response — on success, `p_resp_object` is the current position.

6.5.1. Deviations for protocol version 11

Request flushing and response processing can be deferred.

If `ptype_batch_send` or higher is used, `op_seek_blob` can be batched with Create/Open (and other blob operations) by using the *invalid object* handle 0xFFFF.

6.6. Cancel

Cancels and invalidates the blob handle. If this was a newly created blob, the blob is disposed.

Client

`Int32` — `p_operation`
: Operation code (`op_cancel_blob`)

`Int32` — `p_rlse_object`
: Blob handle

Server

Generic response

6.6.1. Deviations for protocol version 11

Request flushing and response processing can be deferred.

If `ptype_batch_send` or higher is used, `op_cancel_blob` can be batched with Create/Open (and other blob operations) by using the *invalid object* handle 0xFFFF. Though doing this probably makes little sense for `op_cancel_blob`.

6.7. Close

Closes and invalidates the blob handle.

Client

`Int32` — `p_operation`

Operation code (`op_close_blob`)

`Int32` — `p_rlse_object`

Blob handle

Server

Generic response

6.7.1. Deviations for protocol version 11

Request flushing and response processing can be deferred.

If `ptype_batch_send` or higher is used, `op_close_blob` can be batched with Create/Open (and other blob operations) by using the *invalid object* handle 0xFFFF.

Chapter 7. Arrays

7.1. Get slice

Client

`Int32` — `p_operation`
: Operation code (`op_get_slice`)

`Int32` — `p_slc_transaction`
: Transaction handle

`Int64` — `p_slc_id`
: Array handle

`Int32` — `p_slc_length`
: Slice length

`Buffer` — `p_slc_sdl`
: Slice descriptor (SDL)

`Buffer` — `p_slc_parameters`
: Slice parameters (always empty?, needs verification)

`Buffer` — `p_slc_slice`
: Slice data (always empty)

Server

Success response: Slice response

Failure response: Generic response

7.2. Put slice

Client

`Int32` — `p_operation`
: Operation code (`op_put_slice`)

`Int32` — `p_slc_transaction`
: transaction handle

`Int64` — `p_slc_id`
: Array handle

`Int32` — `p_slc_length`

Slice length

`Buffer` — `p_slc_sdl`

Slice descriptor (SDL)

`Buffer` — `p_slc_parameters`

Slice parameters (always empty?, needs verification)

`Buffer`` — `p_slc_slice`

Slice data

Server

Generic response — on success, `p_resp_blob_id` is the array handle.

Chapter 8. Batches

Statement batches were introduced in protocol 16 (Firebird 4.0).

8.1. Create

Client

`Int32` — `p_operation`
: Operation code (`op_batch_create`)

`Int32` — `p_batch_statement`
: Statement handle

`Buffer` — `p_batch_blr`
: BLR format of batch messages

`UInt32` — `p_batch_msglen`
: Message length

`Buffer` — `p_batch_pb`
: Batch parameters buffer

If `ptype_lazy` or higher, flushing and response processing can be deferred.

Server

Generic response

8.2. Send messages

Client

`Int32` — `p_operation`
: Operation code (`op_batch_msg`)

`Int32` — `p_batch_statement`
: Statement handle

`UInt32` — `p_batch_messages`
: Number of messages

`Buffer` — `p_batch_data`
: Batched values (formatted message repeats 'Number of messages' times)

Server

Generic response

8.3. Execute batch

Client

`Int32` — `p_operation`

Operation code (`op_batch_exec`)

`Int32` — `p_batch_statement`

Statement handle

`Int32` — `p_batch_transaction`

Transaction handle

Server

Success response:

`Int32` — `p_operation`

Operation code

If operation equals `op_batch_cs`:

Batch completion state

`Int32` — `p_batch_statement`

Statement handle

`UInt32` — `p_batch_reccount`

Total records count

`UInt32` — `p_batch_updates`

Number of update counters (records updated per each message)

`UInt32` — `p_batch_vectors`

Number of per-message error blocks (message number in batch and status vector of an error processing it)

`UInt32` — `p_batch_errors`

Number of simplified per-message error blocks (message number in batch without status vector)

`Byte[]`

Update counters (records updated per each message), array of `Int32`, length is equal to `p_batch_updates`

Length is `p_batch_updates * 4` bytes long.

`Byte[]`

Detailed info about errors in batch (for each error server sends number of message (`Int32`) and status vector in standard way (exactly like in op_response). Number of such pairs is equal to `p_batch_vectors`.

Length can only be determined by correctly parsing the `<Int32><statusvector>` pairs.

`Byte[]`

Simplified error blocks (for each error server sends number of message (`Int32`) w/o status vector). Used when too many errors took place. Number of elements is equal to `p_batch_errors`.

Length is `p_batch_errors * 4` bytes.

Failure response: Generic response

8.4. Release batch

Client

`Int32` — **`p_operation`**

Operation code (`op_batch_rls`)

`Int32` — **`p_batch_statement`**

Statement handle

Server

Generic response

8.5. Cancel batch

Client

`Int32` — **`p_operation`**

Operation code (`op_batch_cancel`)

`Int32` — **`p_batch_statement`**

Statement handle

Server

Generic response

8.6. Sync batch

Introduced in protocol 17 (Firebird 4.0.2).

Used to force the server to acknowledge previously sent lazy intermediate operations (e.g.

op_batch_msg, op_batch_regblob, op_batch_blob_stream and possibly others).

Client

Int32 — p_operation
: Operation code (op_batch_sync)

Server

Generic response

8.7. Set default blob parameters

Client

Int32 — p_operation
: Operation code (op_batch_set_bpb)

Int32 — p_batch_statement
: Statement handle

Buffer — p_batch_blob_bpb
: Default BLOB parameter buffer

Server

Generic response

8.8. Register existing blob

Client

Int32 — p_operation
: Operation code (op_batch_regblob)

Int32 — p_batch_statement
: Statement handle

Int64 — p_batch_exist_id
: Existing BLOB ID

Int64 — p_batch_blob_id
: Batch temporary BLOB ID

Server

Generic response

8.9. Stream of BLOB data

This description needs further verification and possibly correction. For example, it seems to mix up Buffer and Byte[]. We're also not able to match some fields to the implementation. For example, the repeated "Record length" seems to be absent, or may actually refer to the `p_batch_blob_data` buffer length.

Client

`Int32` — `p_operation`

Operation code (`op_batch_blob_stream`)

`Int32` — `p_batch_statement`

Statement handle

`Buffer[]` — `p_batch_blob_data`

BLOB stream

This stream is a sequence of blob records. Each blob records contains:

`UInt32`

Record length

The following three fields are called **BLOB header**

`Int64`

Batch temporary BLOB ID

`UInt32`

BLOB size

`UInt32`

BLOB parameters buffer size

`Buffer`

BLOB parameters buffer

`Buffer`

BLOB data (length - BLOB size bytes) (*what does this mean?*)

BLOB headers and records in a stream need not match, i.e. one record may contain many BLOBs and BLOB may stretch from one record to next.

Server

Generic response

Chapter 9. Services

9.1. Attach

This is essentially the same as database attach, but with `op_service_attach`.

Client

`Int32` — `p_operation`

Operation code (`op_service_attach`)

`Int32` — `p_atch_database`

Database object id; currently always `0`

`String` — `p_atch_file`

Service name

Current Firebird versions only support one service: `service_mgr`. Since Firebird 3.0, this can also be an empty string (empty buffer) with the same meaning.

The encoding is unspecified, but given the only valid name is either ASCII or empty, use of ASCII or an ASCII-compatible encoding (e.g. UTF-8 or extended ANSI code pages) should always work.

`Buffer` — `p_atch_dpb`

Service parameter buffer

Similar to the database parameter buffer of database attach/create, but using `isc_spb_` tags instead of `isc_dpb_`.

Server

Generic response

9.2. Detach

Client

`Int32` — `p_operation`

Operation code (`op_service_detach`)

`Int32` — `p_rlse_object`

Service manager attachment handle (always `0`)

Server

Generic response

9.3. Start

Client

`Int32` — `p_operation`

Operation code (`op_service_start`)

`Int32` — `p_info_object`

Service manager attachment handle (always `0`)

`Int32` — `p_info_incarnation`

Incarnation of object (`0`)

TODO: Usage and meaning?

`Buffer` — `p_info_items`

Service parameter buffer

Server

Generic response

9.4. Query service

Client

`Int32` — `p_operation`

Operation code (`op_service_info`)

`Int32` — `p_info_object`

Service manager attachment handle

`Int32` — `p_info_incarnation`

Incarnation of object (`0`)

TODO: Usage and meaning?

`Buffer` — `p_info_items`

Service parameter buffer

`Buffer` — `p_info_recv_items`

Requested information items

`Int32` — `p_info_buffer_length`

Requested information items buffer length

Server

Generic response — on success, `p_resp_data` contains the requested information.

Chapter 10. Events

10.1. Connection request

Client

`Int32` — `p_operation`

Operation code (`op_connect_request`)

`Int32` — `p_req_type`

Unused, but always use `P_REQ_async` (1) for backwards compatibility

`Int32` — `p_req_object`

Unused, always use `0`

`Int32` — `p_req_partner`

Unused, always use `0`

Server

Generic response — with on success:

`p_resp_data`

Aux connection information

This is part of the `sockaddr_in` structure.

It is not in XDR format

`Int16`

Socket family (can be ignored)

`Int16`

Aux connection port

Remaining bytes

To be ignored: always use the hostname or IP address of the original connection.

After a successful response, the client needs to create a connection to the specified port.

10.2. Queue events

Client

`Int32` — `p_operation`

Operation code (`op_que_events`)

`Int32` — `p_event_database`

Database handle

`Buffer` — `p_event_items`

Events parameter buffer

`Int32` — `p_event_ast`

Unused, always set `0`

`Int32` — `p_event_arg`

Unused, always set `0`

`Int32` — `p_event_rid`

Local event id

Server

Generic response — on success, `p_resp_object` holds the remote event id.

10.3. Cancel events

Client

`Int32` — `p_operation`

Operation code (`op_cancel_events`)

`Int32` — `p_event_database`

Database handle (always `0`)

`Int32` — `p_event_rid`

Local event id

Server

Generic response

Chapter 11. Reading row data

TODO: Processing row data

Appendix A: External Data Representation (XDR)

The Firebird wire protocol uses XDR for exchange messages between client and server.

Appendix B: Data types

`Int32`

Integer 32-bits

In some cases — e.g. object handles, and *some* lengths — this is actually a 16-bit short encoded as a 32-bit int with the high bits zero.

`UInt32`

Unsigned integer 32-bits

`Int64`

Integer 64-bits

Alternatively, especially for blob and arrays ids, can be interpreted as two `Int32`, a.k.a. a “quad”. Interpretation as a 64-bit integer — even for blob and array ids — is generally simpler, and should not make a difference.

`Buffer`

Composed of

`Int32`

Length of buffer data **without** padding

`Byte[]`

Buffer data

`Byte[]`

Padding of 0 to 3 bytes to align the message to a multiple of 4 (e.g. calculated as `(4 - length) & 3`).

That is, for some `N >= 0`, when the buffer length is:

- `N * 4` bytes → no padding
- `N * 4 + 1` bytes → 3 bytes padding
- `N * 4 + 2` bytes → 2 bytes padding
- `N * 4 + 3` bytes → 1 byte padding

`Byte[]`

An array of bytes

Length follows from another field in the message, from correct parsing of the value, or from other specifics of the message.

`String`

A text string, read or written as a Buffer, encoded in the connection character set or some message or context specific character set

Appendix C: Revision history

Revision History			
0.1	31 May 2004		First draft for review.
0.2	02 Jun 2004		Fixed issues reported by Paul Vinkenoog.
0.3	03 Jun 2004		Added new subsections to the Statements section.
0.4	05 Jun 2004		Fixed issues reported by Paul Vinkenoog.
0.5	06 Jun 2004		Fixed issues reported by Paul Vinkenoog.
0.6	07 Jun 2004		Added events system documentation.
0.7	16 Jun 2004		Modifed document ID to wireprotocol.
0.8	17 Jun 2004		Added two new segmendted lists.
0.9	18 Jun 2004		• Improved segmentedlist usage. • Fixed rendering of important tags.
0.10	19 Jun 2004		Changed rendering of important tags using Paul Vinkenoog fix.
0.11	20 Jun 2004		• Added new segmentedlist. • Updated Statements.Prepare documentation. • Updated Statements.Execute documentation. • Updated Blobs.GetSegment documentation. • Updated Blobs.Seek documentation.
0.12	21 Jun 2004		Updated services information.
0.13	13 Sep 2014		Updated and expanded protocol information
0.14	04 Aug 2020	MR	Conversion to AsciiDoc, minor copy-editing
0.15	26 Dec 2021	AP	Document batch execution

Revision History

0.16	13 Apr 2025	MR	• Added Firebird struct field names to message descriptions for reference • Updated, corrected and expanded field descriptions • Documented `op_put_segment` • Added missing field in `p_sgmt_length` in `op_batch_segments` • Documented protocol 11 batching of operations for blobs • Documented protocol 16 timeout (`p_sqldata_timeout`) for `op_execute`/`op_execute2` • Documented protocol 18 cursor flags (`p_sqldata_cursor_flags`) for `op_execute`/`op_execute2` • Documented protocol 19 inline blob size (`p_sqldata_inline_blob_size`) for `op_execute`/`op_execute2` (but not yet `op_inline_blob`!)