Changes to the Firebird API and ODS

ODS (On-Disk Structure) changes

On-disk structure (ODS) changes include the following:

New ODS number

Firebird 2.5 creates databases with an ODS (On-Disk Structure) version of 11.2.

Maximum page size

The maximum page size remains 16 KB (16384 bytes).

Maximum number of page buffers in cache The maximum number of pages that can be configured for the database cache depends on whether the database is running under 64-bit or 32-bit Firebird:

- 64-bit: 2^31 -1 (2,147,483,647) pages
- 32-bit: 128,000 pages, i.e., unchanged from v.2.1.

API (Application Programming Interface) extensions

Additions to the Firebird API include:

BLOB conversions enabled in the API functions

A. dos Santos Fernandes

Tracker reference CORE-3446.

Conversion either way between BLOBs and other types are enabled in the API functions (XSQLVAR or blr messages).

- BLOBs can now be moved to or from different types in the execute and fetch calls.
- For input parameters, it allows a parameter to be put as a string without the need to create and fill a BLOB from the client side.
- For output (execute or fetch), it will be helpful for an application that knows its data and can describe a BLOB as a string with its maximum length.

Connection strings & character sets

A. dos Santos Fernandes

Previous versions had no way to interoperate with the character set(s) used by the operating system and its file system. Firebird 2.5 has been made “environmentally aware” with regard to the file names of databases, other files and string parameters generally, when accessed through and/or passed in API connection requests. This change significantly improves Firebird's ability to accept and work with file names and other parameters containing characters that are not in the ASCII subset.

Only DPB connections support this feature: In the current implementation, only connections made through the DPB (database parameter block) support this feature. It is not supported for Services API (isc_spb*) functions.

isc_dpb_utf8_filename

The new connection option isc_dpb_utf8_filename has been introduced, to enable Firebird to be specifically informed that the file name or other character item being passed is in the UTF8 (UTF-8) character set. If the option is not used, the character set defaults to the code page of the operating system.

Client-server compatibility

New client, older server

If the client is v.2.5 or newer and it is connecting to a pre-v.2.5 remote server, using the isc_dpb_utf8_filename option causes the client to convert the file name from UTF-8 to the client code page before passing it to the server. It removes the isc_dpb_utf8_filename option from the DPB.

Compatibility is assured when the same code page is being used on both the the client and server stations.

New client, new server, without isc_dpb_utf8_filename

If the client is v.2.5 or newer and it is connecting to a v.2.5 or newer remote server without using the isc_dpb_utf8_filename, the client converts the file name from the OS code page to UTF-8 and inserts the isc_dpb_utf8_filename option into the DPB.

The file name received on the server is not subject to any special treatment. However, unlike older clients, the v.2.5 client may convert the file name automatically and insert the isc_dpb_utf8_filename option into the DPB. Compatibility is guaranteed, regardless, when the host and client are using the same code page.

New client, new server, with isc_dpb_utf8_filename

Whenever the isc_dpb_utf8_filename option is used, the client passes the unmodified file name to the server. The client thus always passes a UTF-8 file name to the server along with the isc_dpb_utf8_filename option.

back to top of page

Code page conversions

On Windows the code page used for conversions is Windows ANSI. On all other platforms, UTF-8 is used.

The operating system code page and UTF-8 may not be the best choice for file names. For example, if you had a script or other text file for processing in isql or some other script-running tool that used another connection character set, it would not be possible to edit the file correctly using multiple character sets (code pages).

There is a solution: the Unicode code point. If used correctly, it enables correct interpretation of a character even if the client is older than v.2.5.

Using Unicode code points

Any Unicode character may now now be encoded on the connection string file name as though it were an ASCII character. It is accomplished by using the symbol # as a prefix for a Unicode code point number (in hexadecimal format, similar to U+XXXX notation).

Write it as #XXXX with X being 0-9, a-f, A-F.

If one of the characters happens to be the literal #, you could either “double” the hash character (##) or use the code point number for it, #0023.

Note: The hash character is interpreted at the server with these new semantics, even if the client is older than v.2.5.

Support for SQLSTATE completion codes

W. Oliver

D. Yemanov

Tracker reference CORE-1761.

A new client-side API function, fb_sqlstate() is available to convert the status vector item for an error into the corresponding SQL-2003 standard 5-alphanumeric SQLSTATE.

- The SQLSTATE code represents the concatenation of a 2-character SQL CLASS and a 3-character SQL SUBCLASS.
- Statements now return an SQLSTATE completion code.
- The isql utility now prints the SQLSTATE diagnostic for errors instead of the SQLCODE one
- The SQLCODE diagnostic is deprecated—meaning it will disappear in a future release.
- (v.2.5.1) Exception handling syntax in PSQL, of the style WHEN SQLSTATE, has been added.

Deprecated SQLCODE: Although the SQLCODE is deprecated and use of the SQLSTATE is preferred, it remains in Firebird for the time being. The isc_sqlcode() API function is still supported, as is the WHEN SQLCODE exception handling.

Appendix A: SQLSTATE provides a list of all SQLSTATE codes in use in this release, along with the corresponding message texts.

back to top of page

"Efficient unprepare"

W. Oliver

D. Yemanov

Tracker reference CORE-1741.

The new option DSQL_unprepare (numeric value 4) for the API routine isc_dsql_free_statement() allows the DSQL statement handle to survive the “unpreparing” of the statement.

Previously, the isc_dsql_free_statement() function supported only DSQL_close (for closing a named cursor) and DSQL_drop (which frees the statement handle).

The API addition is:

```
#define DSQL_close 1
#define DSQL_drop 2
#define DSQL_unprepare 4
```

Cancel operation function

Alex Peshkov

New fb_cancel_operation() API call, allowing cancellation of the current activity being performed by some kind of blocking API call in the given connection.

Syntax

```
 ISC_STATUS fb_cancel_operation(ISC_STATUS* status_vector,
                             isc_db_handle* db_handle,
                             ISC_USHORT option);
```

Parameters

- **status vector (ISC_STATUS* status_vector)**: A regular status vector pointer structure.
- **db_handle (pointer to a isc_db_handle)**: A regular, valid database handle. It identifies the attachment.
- **option (unsigned short: symbol)**: Determines the action to be performed. The option symbols are:
    - fb_cancel_raise: cancels any activity related to the db_handle specified in the second parameter. The effect will be that, as soon as possible, the engine will try to stop the running request and return an exception to the caller via the status vector of the interrupted API call.

“..as soon as possible” will be, under normal conditions, at the next rescheduling point.

Example

```
       Thread1:                              Thread2:
       ------------------------------        ------------------------------
```

isc_dsql_execute (status, ...)

```
       ........                              fb_cancel_operation(cancel_status,
...)
       status[1] == isc_cancelled;           cancel_status[1] = 0;
```

- fb_cancel_disable: disables execution of fb_cancel_raise requests for the specified attachment. It can be useful when your program is executing critical operations, such as cleanup, for example.
- fb_cancel_enable: re-enables delivery of a cancel execution that was previously disabled. The 'cancel' state is effective by default, being initialized when the attachment is created.
- fb_cancel_abort: : forcibly close client side of connection. Useful if you need to close a connection urgently. All active transactions will be rolled back by the server. 'Success' is always returned to the application. Use with care!

Usage

The cycle of fb_cancel_disable and fb_cancel_enable requests may be repeated as often as necessary. If the engine is already in the requested state there is no exception: it is simply a no-op.

Usually fb_cancel_raise is called when you need to stop a long-running request. It is called from a separate thread, not from the signal handler, because it is not async signal safe.

Pay attention to the asynchronous nature of this API call!

Another aspect of asynchronous execution is that, at the end of API call, the attachment's activity might be cancelled or it might not. The latter is always a possibility. The asynchronicity also means that returned status vector will almost always return FB_SUCCESS. Exceptions, though, are possible: a network packet error, for example.

Example

```
Thread A:
fb_cancel_operation(isc_status, &DB, fb_cancel_enable);
isc_dsql_execute_immediate(isc_status, &DB, &TR, 0, "long running
statement", 3, NULL);
// waits for API call to finish...

  Thread B:
  fb_cancel_operation(local_status, &DB, fb_cancel_raise);

Thread A:
```

```
if (isc_status[1])
  isc_print_status(isc_status); // will print "operation was cancelled"
```

back to top of page

Shutdown functions

Alex Peshkov

This release exposes a variety of API functions for instigating server shutdowns of various types from client applications.

Two interrelated fb_shutdown* functions This release exposes two fb_shutdown* functions that may be useful for embedded server applications: fb_shutdown() and fb_shutdown_callback.

Prototypes

typedef int (*FB_SHUTDOWN_CALLBACK)(const int reason, const int mask, void* arg);

int fb_shutdown(unsigned int timeout,

```
               const int reason);
```

ISC_STATUS fb_shutdown_callback(ISC_STATUS* status_vector,

```
                                FB_SHUTDOWN_CALLBACK callback_function,
                                const int mask,
                                void* arg);
```

back to top of page fb_shutdown() fb_shutdown() performs a smart shutdown of various Firebird subsystems (yValve, engine, redirector). It was primarily designed for use by the internal engine, since it is only applicable to the current process. It is exposed by the API for its possible usefulness to user applications in the embedded server environment.

Currently operational only for the embedded engine, this function terminates all the current activity, rolls back active transactions, disconnects active attachments and shuts down the embedded engine instance gracefully.

Important for application developers: fb_shutdown() does not perform a shutdown of a remote server to which your application might be concurrently attached. In fact, all of the Firebird client libraries — including the one in embedded — call it automatically at exit(), as long as the client is attached to at least one database or service.

Hence, it should never be called by a client in the context of a remote attachment.

Parameters

fb_shutdown() takes two parameters:

timeout in milliseconds reason for shutdown The reason codes (const int reason), which are negative, are listed in ibase.h: refer to constants starting with fb_shutrsn. Note: when calling fb_shutdown()

from your program, you must pass the value as positive, for it will be passed as an argument to fb_shutdown_callback() by way of your callback_function, the routine where you would code the appropriate actions.

Return Values

A return value of zero means shutdown was successful. A non-zero value means some errors occurred during the shutdown. Details will be written to firebird.log. back to top of page fb_shutdown_callback() fb_shutdown_callback() sets up the callback function that is to be called during shutdown. It is a call that almost always returns successfully, although there are cases, such as an out-of-memory condition, which could cause it to return an error.

Parameters

fb_shutdown_callback() takes four parameters:

status vector (ISC_STATUS* status_vector): A regular status vector pointer structure.

pointer to callback function (FB_SHUTDOWN_CALLBACK callback_function): This points to the callback function you have written to perform the actions (if any) to be taken when the callback occurs.

Your callback function can take three parameters. The first and second parameters help to determine what action is to be taken in your callback:

1. Reason for shutdown

Two shutdown reasons are of especial interest: fb_shutrsn_exit_called: Firebird is closing due to exit() or unloaded client/embedded library. fb_shutrsn_signal, applies only to POSIX: a SIGINT or SIGTERM signal was caught. Note: Firebird code always uses negative reasons. Users are expected to use positive values when calling fb_shutdown() themselves. 2. Actual value of the mask with which it was called

The purpose of this parameter to help determine whether the callback was invoked before or after engine shutdown. 3. Argument passed to fb_shutdown_callback() by the user application

Can be used for any purpose you like and may be NULL.

Return value from the callback function

If the callback function returns zero, it means it performed its job successfully. A non-zero return value is interpreted according to the call mask (see next parameter topic, below):

For fb_shut_postproviders calls, it means some errors occurred and it will result in a non-zero value being returned from fb_shutdown(). It is the responsibility of the callback function to notify the world of the exact reasons for the error condition being returned. For fb_shut_preproviders calls, it means that shutdown will not be performed. Tip: It is NOT a good idea to return non-zero if the shutdown is due to exit() having been called! 😉

call mask (const int mask): Can have the following symbolic values:

fb_shut_preproviders: callback function will be called before shutting down engine. fb_shut_postproviders: callback function will be called after shutting down engine. An ORed combination of them, to have the same function called in either case. Values for call mask

fb_shut_confirmation: Engine queries: Is everyone ready to shut down? fb_shut_preproviders: Actions to be done before providers are closed. fb_shut_postproviders: Actions to be done when providers are already closed. fb_shut_finish: Final cleanup.

Returning non-zero for fb_shut_confirmation (although not fb_shut_preproviders) means that shutdown will not be performed.

argument (void* arg): This is the argument to be passed to callback_function.

back to top of page Using the fb_shutdown functions Following is a sample of using the shutdown and shutdown callback feature to prevent your program from being terminated if someone presses [Ctrl + C] while it is has database attachments.

#include <ibase.h>

callback function for shutdown static int ignoreCtrlC(const int reason, const int, void) { return reason == fb_shutrsn_signal ? 1 : 0; } int main(int argc, char *argv[]) { ISC_STATUS_ARRAY status; if (fb_shutdown_callback(status, ignoreCtrlC, fb_shut_confirmation, 0)) { isc_print_status(status); return 1; }* your code continues … } back to top of page New isc_spb_prp_* constants for shutdown The new database shutdown modes can now be set using calls to the Services API. A number of new isc_spb_prp_* constants are available as arguments.

isc_spb_prp_shutdown_mode and isc_spb_prp_online_mode These arguments are used for shutting down a database and bringing it back on-line, respectively. Each carries a single-byte parameter to set the new shutdown mode, exactly in accord with the gfix -shut settings:

isc_spb_prp_sm_normal isc_spb_prp_sm_multi isc_spb_prp_sm_single isc_spb_prp_sm_full The shutdown request also requires the type of shutdown to be specified, viz., one of

isc_spb_prp_force_shutdown isc_spb_prp_attachments_shutdown isc_spb_prp_transactions_shutdown Each takes a 4-byte integer parameter, specifying the timeout for the shutdown operation requested.

Note: The older-style parameters are also supported and should be used to enter the default shutdown (currently 'multi') and online ('normal') modes.

Usage Examples

Following are a few examples of using the new parameters with the fbsvmgr utility. For simplicity, it is assumed that login has already been established. Each example, though broken to fit the page-width, is a single line command.

Shutdown database to single-user maintenance mode:

fbsvcmgr service_mgr action_properties dbname employee

```
prp_shutdown_mode prp_sm_single prp_force_shutdown 0
```

Next, enable multi-user maintenance:

fbsvcmgr service_mgr action_properties dbname employee

```
prp_online_mode prp_sm_multi
```

Now go into full shutdown mode, disabling new attachments for the next 60 seconds:

fbsvcmgr service_mgr action_properties dbname employee

```
prp_shutdown_mode prp_sm_full prp_attachments_shutdown 60
```

Return to normal state:

fbsvcmgr service_mgr action_properties dbname employee

```
prp_online_mode prp_sm_normal
```

back to top of page Tighter control over header-level changes Alex Peshkov

Several DPB parameters have been made inaccessible to ordinary users, closing some dangerous loopholes. In some cases, they are settings that would alter the database header settings and potentially cause corruptions if not performed under administrator control; in others, they initiate operations that are otherwise restricted to the SYSDBA. They are:

isc_dpb_shutdown and isc_dpb_online isc_dpb_gbak_attach, isc_dpb_gfix_attach and isc_dpb_gstat_attach isc_dpb_verify isc_dpb_no_db_triggers isc_dpb_set_db_sql_dialect isc_dpb_sweep_interval isc_dpb_force_write isc_dpb_no_reserve isc_dpb_set_db_readonly isc_dpb_set_page_buffers (on Superserver) The parameter isc_dpb_set_page_buffers can still be used by ordinary users on Classic and it will set the buffer size temporarily for that user and that session only. When used by the SYSDBA on either Superserver or Classic, it will change the buffer count in the database header, i.e., make a permanent change to the default buffer size.

Important note for developers and users of data access drivers and tools: This change will affect any of the listed DPB parameters that have been explicitly set, either by including them in the DPB implementation by default property values or by enabling them in tools and applications that access databases as ordinary users. For example, a Delphi application that included 'RESERVE PAGE SPACE=TRUE' and 'FORCED WRITES=TRUE' in its database Params property, which caused no problems when the application connected to Firebird 1.x, 2.0.1. 2.0.3, 2.04 or 2.1.0/2.1.1, now rejects a connection by a non-SYSDBA user with ISC ERROR CODE 335544788, “Unable to perform operation. You must be either SYSDBA or owner of the database.”

back to top of page New trace services for applications Vlad Khorsun

Five new services relating to the management of the new user trace sessions have been added to the Services Manager, each with its corresponding Services API action function.

isc_action_svc_trace_start Starts a user trace session.

Parameter(s)

isc_spb_trc_name : trace session name, string, optional isc_spb_trc_cfg : trace session configuration, string, mandatory The mandatory parameter is a string encompassing the text for the desired configuration. A template file named fbtrace.conf is provided in Firebird's root directory as a guide to the contents of this string.

Note: 1. Unlike system audit sessions, a user session does not read the configuration from a file. It will be the responsibility of the application developer to devise a mechanism for storing configurations locally at the client and retrieving them for run-time use. 2. Superfluous white space in the string is fine: it will simply be ignored.

Output

A text message reporting the status of the operation, EITHER: Can not start trace session. There are no trace plugins loaded OR

Trace session ID NNN started

In the second case, the results of the trace session in text format follow.

isc_action_svc_trace_stop Stops a designated trace session.

Parameter(s)

isc_spb_trc_id : trace session ID, integer, mandatory Output

A text message providing the result (status) of the request:

Trace session ID NNN stopped. No permissions to stop other user trace session. Trace session ID NNN not found. back to top of page isc_action_svc_trace_suspend Suspends a designated trace session.

Parameter(s)

isc_spb_trc_id : trace session ID, integer, mandatory Output

A text message providing the result (status) of the request:

Trace session ID NNN paused. No permissions to change other user trace session. Trace session ID NNN not found. back to top of page isc_action_svc_trace_resume Resumes a designated trace session that has been suspended.

Parameter(s)

isc_spb_trc_id : trace session ID, integer, mandatory Output

A text message providing the result (status) of the request:

Trace session ID NNN resumed. No permissions to change other user trace session. Trace session ID NNN not found.

isc_action_svc_trace_list Lists existing trace sessions.

No parameters.

Output

A text message listing the trace sessions and their states:

Session ID: <number>. name: <string>. Prints the trace session name if it is not empty. user: <string>. Prints the user name of the user that created the trace session. date: YYYY-MM-DD

HH:NN:SS, start date and time of the user session. flags: <string>, a comma-separated set comprising some or all of the following: active | suspend: Run state of the session. admin: Shows admin if an administrator user created the session. Absent if an ordinary user created the session. system: Shows system if the session was created by the Firebird engine (system audit session). Absent if an ordinary user created the session. audit | trace: Indicates the kind of session: audit for an engine-created audit session or trace for a user trace session. log full: Conditional, appears if it is a user trace session and the session log file is full. Note: The output of each service can usually be obtained using a regular isc_service_query call with either of the isc_info_svc_line or isc_info_svc_to_eof information items.

back to top of page Back up to or restore from a remote backup file Alex Peshkov

Introduced in v.2.5.2 was the ability to run gbak at the server side reading from or writing to a gbak backup file located at a remote client. This is an especially efficient way to do backups and restores across an Internet connection.

The simplest way to use this feature is with the command-line tool fbsvcmgr which provides a quick-and-dirty interface for Services API calls without the need to write your own client application.

Remote backup To back up a remote database using fbsvcmgr, enter a command with the following pattern:

fbsvcmgr remotehost:service_mgr -user sysdba -password XXX \

```
action_backup -dbname some.fdb -bkp_file stdout >some.fbk
```

Restore from a remote backup file To restore a database from a remotely located backup file using fbsvcmgr, enter a command with the following pattern:

fbsvcmgr remotehost:service_mgr -user sysdba -password XXX \

```
action_restore -dbname some.fdb -bkp_file stdin <some.fbk
```

Note: The “verbose” (-v[erify]) switch cannot be used when performing backup because the data channel from server to client is used to deliver blocks of data from the backup file. If you try to use it you will get an error message.

When restoring a database, verbose mode may be used without limitations.

Writing your own utility code If you want to perform a remote backup or restore from your own program, use the Services API.

Backup Backup is very simple - Data, returned by repeating calls to isc_service_query() tagged with isc_info_svc_to_eof, are a stream representing an image of the backup file. Just pass “stdout” to the server as the backup file name, using isc_info_svc_to_eof tag in the isc_service_query() call.

Restore Restore is not so straightforward. The client sends the new spb parameter isc_info_svc_stdin to the server in the isc_service_query() call. If the service needs some data in stdin, it returns isc_info_svc_stdin in the query results, followed by 4-byte value representing the number of bytes it is ready to accept from client. (Zero value means no more data is needed right now.)

Important: The client should not send more data than is requested by the server: it will throw the error Size of data is more than requested.

Data is sent in the next isc_service_query() call, in the send_items block, using the traditional form of the isc_info_svc_line tag: isc_info_svc_line, 2 bytes length, data. When the server needs the next portion, it reverts to returning a non-zero value for isc_info_svc_stdin from isc_service_query().

Tip: A sample of using the Services API for remote backup and restore can be found in the source code of fbsvcmgr.

back to top of page Other Services API additions Alex Peshkov

Other additions to the Services API include:

Mapping for RDB$ADMIN role in the Services API Two tag items have been added to the services parameter block (SPB) to to enable or disable the RDB$ADMIN role for a privileged operating system user when requesting access to the security database.

Note: This capability is implemented in the gsec utility by way of the new -mapping switch.

Refer to the notes in the relevant section of the Command-line utilities chapter.

Tag item isc_action_svc_set_mapping Enables the RDB$ADMIN role for the appointed OS user for a service request to access security2.fdb.

Tag item isc_action_svc_drop_mapping Disables the RDB$ADMIN role for the appointed OS user for a service request to access security2.fdb.

Parameter isc_spb_sec_admin The new parameter isc_spb_sec_admin, is the SPB implementation of the new DDL syntax introduced to enable SYSDBA or another sufficiently privileged user to grant or revoke the RDB$ADMIN role in the security database (security2.fdb) to or from an ordinary Firebird user. An ordinary user needs this role to acquire the same privileges as SYSDBA to create, alter or drop users in the security database.

isc_spb_sec_admin is of type spb_long with a value of either 0 (meaning REVOKE ADMIN ROLE) or a nonzero number (meaning GRANT ADMIN ROLE).

For more information, refer to the topic CREATE/ALTER/DROP USER in the chapter Data Definition Language.

Tag item isc_spb_bkp_no_triggers This new SPB tag reflects the Services API side of the -nodbtriggers switch introduced in the gbak utility in v.2.1 to prevent database-level and transaction-level triggers from firing during backup and restore. It is intended for use as a member of the isc_spb_options set of optional directives that includes items like isc_spb_bkp_ignore_limbo, etc.

Tag item isc_spb_res_metadata_only Tracker reference: CORE-3462.

If you pass the isc_spb_bkp_metadata_only tag to the restore service option in the SPB, it will perform a metadata-only restore. Many tools, including Firebird's fbsvcmgr do not provide for specifying a backup option in a restore request.

(v.2.5.1) To avoid confusion, the tag isc_spb_res_metadata_only, simply reimplementing isc_spb_bkp_metadata_only was added as a constant to the public header and fbsvcmgr.

back to top of page nBackup support Three additions were made to support nBackup actions in the Services API.

Backup and restore Tracker reference: CORE-1758.

The nBackup utility performs two logical groups of operations: locking or unlocking a database and backing it up or restoring it. While there is no rationale for providing a service action for the lock/unlock operations — they can be requested remotely by way of an SQL language request for ALTER DATABASE — a Services API interface to the backup/restore operations is easily justified.

Backup and restore must be run on the host station and the only way to access them was by running nBackup.

The two new service actions now enabling nBackup backup and restore to be requested through the Services API are:

isc_action_svc_nbak - incremental nbackup. isc_action_svc_nrest - incremental database restore. The parameter items are:

isc_spb_nbk_level - backup level (integer). isc_spb_nbk_file - backup file name (string). isc_spb_nbk_no_triggers - option to suppress database triggers. Usage examples

The following are a few examples of using the new parameters with the fbsvcmgr utility. For simplicity, it is assumed that login has already been established. Each example, though broken to fit the page-width, is a single line command.

Create backup level 0:

fbsvcmgr service_mgr action_nbak dbname employee

```
nbk_file e.nb0 nbk_level 0
```

Create backup level 1:

fbsvcmgr service_mgr action_nbak dbname employee

```
nbk_file e.nb1 nbk_level 1
```

Restore database from those files:

fbsvcmgr service_mgr action_nrest dbname e.fdb

```
nbk_file e.nb0 nbk_file e.nb1
```

back to top of page Direct I/O feature support isc_spb_nbk_direct on|off

This new tag enables the same action as calling nbackup -D on|off from the command line, to force direct I/O on or off. As with other nbackup-related tags, it is valid only with action_nbak.

Note: Information regarding the usage of this feature can be found in the notes about the new nbackup switches in the Utilities chapter.

back to top of page FIX_FSS_DATA and FIX_FSS_METADATA options enabled in Services API A. Peshkov

Tracker reference CORE-2439.

The FIX_FSS_DATA and FIX_FSS_METADATA functions that were implemented as gbak -restore switches in Firebird 2.1 have been implemented in the core engine and exposed as corresponding tag constants for the isc_action_svc_restore structure in the Services API. Thus, developers now have a path for writing applications to automate the migration of older Firebird databases to the new on-disk structure.

The new SPB tags are isc_spb_res_fix_fss_data and isc_spb_res_fix_fss_metadata.

back to top of page New Trace API A new Trace API is under construction, providing a set of hooks which can be implemented as an external plugin module to be called by the engine when any traced event happens. It is as yet undocumented, since it is subject to change in forthcoming sub-releases.

For a little more information about it, refer to the topic Trace plug-in facilities in the Administrative features chapter.