IBPhoenix trace plugin and utilities

Release 1.0

Pavel Císař, IBPhoenix

Jan 12, 2023

CONTENTS:

CHAPTER

ONE

INTRODUCTION

The ability to trace events occurring in the Firebird server was one of the most significant new features of version 2.5. In subsequent versions, this functionality was further improved. Tracing makes it possible not only to diagnose the causes of problems (mainly with performance), but also to continuously monitor server activity and actively prevent problems from occurring.

Despite these indisputable advantages, the tracing options are only used to a very limited extent by Firebird users. The main reason is the fact that the standard trace plugin supplied as part of Firebird produces a text description of events in a stable but very loose and verbose format intended primarily for visual presentation.

But analyzing the events from the trace directly by the user is only possible if the number of events is very small, and is completely unsuitable for monitoring traffic and preventing problems. Only machine event processing is suitable for such purposes.

Although the output from standard trace plugin can also be processed by machine, such an undertaking is complicated (requires a complex parser), time-consuming and, despite all efforts, its efficiency will be very low.

Therefore, IBPhoenix designed and created a completely new trace plugin that is optimized for machine processing of trace events.

1.1 The main advantages of the new plugin:

- Possibility of very detailed and precise setting of monitored events and provided information.
- Event data provided in JSON or Google protocol buffer (protobuf) format.
- The same applies to the trace session configuration.
- Optimization for the speed and efficiency of collection and transmission of trace data.

CHAPTER

TWO

TRACE PLUGIN

2.1 Installation

The plugin is a direct replacement for the standard plugin that comes with Firebird, and its installation is just as simple and follows the same rules as installing a standard plugin.

The plugin comes as a single dynamic library for the 64-bit version of Firebird 3+ (*libibptrace.so* for Linux and *ibptrace.dll* for Windows) that needs to be copied into the *plugins* directory of the Firebird installation.

Subsequently, the *TracePlugin* item in the *firebird.conf* file needs to be modified as follows:

```
TracePlugin = ibptrace
```

The change in settings will take effect only after the Firebird server is restarted.

No additional modifications are required to use the plugin in user trace mode. For use in audit mode, it is also necessary to edit the *AuditTraceConfigFile* entry in the *firebird.conf* file.

2.2 Basic characteristics

The plugin uses data structures in the form of Google protocol buffer (protobuf) for trace event representation and trace configuration. Since Firebird supports only text output from user trace session, and configuration in text format, the plugin uses the textual form of protobuf either in the form of JSON representation (directly supported by libraries for protobuf) or binary form of protobuf messages encoded as BASE64 text.

The format used for the trace session configuration is automatically detected by the plugin, but the output format must be specified within the trace session configuration.

The protobuf format coded as BASE64 is more compact and therefore more economical for transferred data, and is optimal for further processing using the protobuf format. The JSON format is significantly less efficient, but is suitable for processing by a wide variety of tools and programming languages that support this format, and can also be used for visual inspection by the user.

Important: Each individual line of text output from the plugin represents one event, regardless of the output format used.

The plugin distribution includes files *fbtrace.proto* and *fbtraceconf.proto* containing protobuf (proto3 version) definitions of data messages and types used by the plugin. The toolkit supplied with the plugin also includes modules for working with these data structures in Python. To directly use these structures in other languages (C++, C#, Java, Go, Kotlin or Dart) it is necessary to use the protocol buffer compiler and appropriate API.

Tip: Beside languages directly supported by Google, you may use other implementations of protocol buffers.

Alternatively, you can work directly with the JSON representation of these data structures.

2.3 Configuration

Because the configuration of the trace plugin allows very detailed and precise setting of monitored events and provided information, it uses a different mechanism than the standard trace plugin.

While the configuration of a standard trace plugin has only two structural blocks (for the database and for the service) that have a flat structure with parameters at the same level, the configuration of the IBPhoenix trace plugin is represented by a single protobuf message of the *TraceConfig* type.

The configuration is structured into logical units, represented by *GeneralFilter* message and individual messages of the *TopicSpec* type as individual *TraceConfig* items.

Note: A *GUI configuration editor* is available for common use, working with JSON or BASE64 format of saved configuration.

2.3.1 Configuration parameters

The following basic overview of configuration parameters uses dot notation for simplicity as when accessing individual items of a configuration object.

In JSON format, each dot-separated name represents a key in the dictionary. For example parameter names:

```
trace.database_name
db_events.trace.enabled
db_events.trace.event
db_events.output.att_info
```

are translated to:

```
{
 "trace": {
   "database_name": ".*"
 },
 "db_events": {
   "trace": {
     "enabled": true,
     "event": ["create","drop"],
   },
   "output": {
     "att_info": 1,
   }
 }
}
```

Note: Field type marked as **repeated** means that given field could be repeated zero or more times. The order of the repeated values will be preserved.

In JSON format (and generated code for particular programming language), the repeated fields are represented as list of values (of particular data type).

Important: Please note that each dictionary present in the configuration in JSON format may not contain all available parameters (keys). However, missing values are treated as if they were included with the default value (it's how proto3 format works).

Table 1: General filters

Name	Data type	Description
trace.database_name	string	regex for database name
trace.remote_process_name	string	regex for remote process name
trace.attachment_id	uint64	Trace single attachment with specific ID
trace.remote_pid	uint64	Trace all attachments from single remote process specified by PID
trace.remote_address	string	Trace all attachments/processes from single host
trace.user_name	string	Trace all attachments from specified user
trace.role_name	string	Trace all attachments for specific role

Note: General filters are applied when the plugin decides whether or not to trace a particular connection. It's possible to set one or more options. If more options are set, the logic operation between them is AND.

Table 2: Database events

Name	Data type	Description
db_events.trace.enabled	bool	Trace database attachment events or not
db_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **create**, **drop**, **attach**, **detach**
db_events.trace.result	repeated *Result*	If specified, emit only for specified results
db_events.output.att_info	*OutputFilter*	When database attachment details are included

Table 3: Transaction events

Name	Data type	Description
tra_events.trace.enabled	bool	Trace transaction events or not
tra_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **start**, **end**
tra_events.trace.result	repeated *Result*	If specified, emit only for specified results
tra_events.trace.threshold	uint64	If specified, emit only when action is slower than threshold (ms)
tra_events.output.att_info	*OutputFilter*	When database attachment details are included
tra_events.output.tra_info	*OutputFilter*	When transaction details are included
tra_events.output.perf.time_count	sint32	When total time and rows affected are included, see *Performance filter*
tra_events.output.perf.page_io	sint32	When page I/O stats are included, see *Performance filter*
tra_events.output.perf.row_io	sint32	When row I/O stats are included, see *Performance filter*

Table 4: Stored procedure events

Name	Data type	Description
proc_events.trace.enabled	bool	Trace procedure events or not
proc_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **start**, **end**
proc_events.trace.result	repeated *Result*	If specified, emit only for specified results
proc_events.trace.threshold	uint64	If specified, emit only when action is slower than threshold (ms)
proc_events.output.att_info	*OutputFilter*	When database attachment details are included
proc_events.output.tra_info	*OutputFilter*	When transaction details are included
proc_events.output.inputs	sint32	Include input parameters: -1 = all, 0 = none, *n* = up to <n> parameters
proc_events.output.perf.time_count	sint32	When total time and rows affected are included, see *Performance filter*
proc_events.output.perf.page_io	sint32	When page I/O stats are included, see *Performance filter*
proc_events.output.perf.row_io	sint32	When row I/O stats are included, see *Performance filter*
proc_events.output.items	Struct	Key/value pairs for event-specific output items. Valid keys are: • **inputs** (sint32) - WHEN include input params (if proc_events.output.inputs != 0): – -1 = if both start & end are traced, then only for start; if only start or end is traced, then it's like always – 0 = allways (for both start and end events) – >0 = only for end event when execution is slower than value (ms)

Table 5: Stored function events

Name	Data type	Description
func_events.trace.enabled	bool	Trace function events or not
func_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **start**, **end**
func_events.trace.result	repeated *Result*	If specified, emit only for specified results
func_events.trace.threshold	uint64	If specified, emit only when action is slower than threshold (ms)
func_events.output.att_info	*OutputFilter*	When database attachment details are included
func_events.output.tra_info	*OutputFilter*	When transaction details are included
func_events.output.inputs	sint32	Include input parameters: -1 = all, 0 = none, *n* = up to <n> parameters
func_events.output.perf.time_count	sint32	When total time and rows affected are included, see *Performance filter*
func_events.output.perf.page_io	sint32	When page I/O stats are included, see *Performance filter*
func_events.output.perf.row_io	sint32	When row I/O stats are included, see *Performance filter*
func_events.output.items	Struct	Key/value pairs for event-specific output items. Valid keys are: • **inputs** (sint32) - WHEN include input params (if func_events.output.inputs != 0): – -1 = if both start & end are traced, then only for start; if only start or end is traced, then it's like always – 0 = allways (for both start and end events) – >0 = only for end event when execution is slower than value (ms)

Table 6: Trigger events

Name	Data type	Description
trigger_events.trace.enabled	bool	Trace trigger events or not
trigger_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **start**, **end**
trigger_events.trace.result	repeated *Result*	If specified, emit only for specified results
trigger_events.trace.threshold	uint64	If specified, emit only when action is slower than threshold (ms)
trigger_events.output.att_info	*OutputFilter*	When database attachment details are included
trigger_events.output.tra_info	*OutputFilter*	When transaction details are included
trigger_events.output.perf.time_count	sint32	When total time and rows affected are included, see *Performance filter*
trigger_events.output.perf.page_io	sint32	When page I/O stats are included, see *Performance filter*
trigger_events.output.perf.row_io	sint32	When row I/O stats are included, see *Performance filter*

Table 7: Assignment to context variables

Name	Data type	Description
ctx_events.trace.enabled	bool	Trace context variable assignments or not
ctx_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **set**
ctx_events.trace.filter	Struct	Key/value pairs for event-specific filters. Valid keys are: • **namespace** (string) - emit only for specified namespace • **name** (string) - emit only for specified variable name
ctx_events.output.att_info	*OutputFilter*	When database attachment details are included
ctx_events.output.tra_info	*OutputFilter*	When transaction details are included

Table 8: DSQL statement lifecycle events

Name	Data type	Description
dsql_events.trace.enabled	bool	Trace DSQL statements or not
dsql_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **prepare**, **start**, **end**, **free**
dsql_events.trace.result	repeated *Result*	If specified, emit only for specified results
dsql_events.trace.threshold	uint64	If specified, emit only when action is slower than threshold (ms)
dsql_events.trace.filter	Struct	Key/value pairs for event-specific filters. Valid keys are: • **sql.include** (string) - emit ONLY when SQL contains text • **sql.exclude** (string) - DO NOT emit when SQL contains text
dsql_events.output.att_info	*OutputFilter*	When database attachment details are included
dsql_events.output.tra_info	*OutputFilter*	When transaction details are included
dsql_events.output.inputs	sint32	Include input parameters: -1 = all, 0 = none, *n* = up to <n> parameters
dsql_events.output.perf.time_count	sint32	When total time and rows affected are included, see *Performance filter*
dsql_events.output.perf.page_io	sint32	When page I/O stats are included, see *Performance filter*
dsql_events.output.perf.row_io	sint32	When row I/O stats are included, see *Performance filter*
dsql_events.output.items	Struct	Key/value pairs for event-specific output items. Valid keys are: • **text** (bool) - include SQL text • **plan** (bool) - include standard execution plan • **explained** (bool) - include explained execution plan • **inputs** (sint32) - WHEN include input params (if dsql_events.output.inputs != 0): – -1 = if both start & end are traced, then only for start; if only start or end is traced, then it's like always – 0 = allways (for both start and end events) – >0 = only for end event when execution is slower than value (ms)

Table 9: BLR requests

Name	Data type	Description
blr_events.trace.enabled	bool	Trace BLR requests or not
blr_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **compile**, **exec**
blr_events.trace.result	repeated *Result*	If specified, emit only for specified results
blr_events.trace.threshold	uint64	If specified, emit only when action is slower than threshold (ms)
blr_events.output.att_info	*OutputFilter*	When database attachment details are included
blr_events.output.tra_info	*OutputFilter*	When transaction details are included
blr_events.output.perf.time_count	sint32	When total time and rows affected are included, see *Performance filter*
blr_events.output.perf.page_io	sint32	When page I/O stats are included, see *Performance filter*
blr_events.output.perf.row_io	sint32	When row I/O stats are included, see *Performance filter*
blr_events.output.items	Struct	Key/value pairs for event-specific output items. Valid keys are: • **data** (bool) - include BLR data • **text** (bool) - include BLR text Note that BLRInfo is not included at all if neither 'data' or 'text' is provided

Table 10: DYN requests

Name	Data type	Description
dyn_events.trace.enabled	bool	Trace DYN requests or not
dyn_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **exec**
dyn_events.trace.result	repeated *Result*	If specified, emit only for specified results
dyn_events.trace.threshold	uint64	If specified, emit only when action is slower than threshold (ms)
dyn_events.output.att_info	*OutputFilter*	When database attachment details are included
dyn_events.output.tra_info	*OutputFilter*	When transaction details are included
dyn_events.output.items	Struct	Key/value pairs for event-specific output items. Valid keys are: • **data** (bool) - include DYN data • **text** (bool) - include DYN text

Table 11: Service events

Name	Data type	Description
svc_events.trace.enabled	bool	Trace service events or not
svc_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **attach**, **start**, **query**, **detach**
svc_events.trace.result	repeated *Result*	If specified, emit only for specified results
svc_events.trace.filter	Struct	Key/value pairs for event-specific filters. Valid keys are: • **query.include_fetch_output** (bool) - when False, does not emit TraceEntry when *recv_items* contain *isc_info_svc_line* or *isc_info_svc_to_eof* • **svc.include** (string) - emit ONLY when service name matches given regex • **svc.exclude** (string) - DO NOT emit when service name matches given regex
svc_events.output.svc_info	*OutputFilter*	When service details are included
svc_events.output.items	Struct	Key/value pairs for event-specific output items. Valid keys are: • **query.sent** (string) - Copy data to *ServiceQuery* message • **query.received** (string) - Copy data to *ServiceQuery* message Valid values: • **all** - Copy the whole value into sent/received field of *ServiceQuery* message • **tags** - Copy only *isc_info_** tags into sent/received field of *ServiceQuery* message

Table 12: Errors and Warnings

Name	Data type	Description
error_events.trace.enabled	bool	Trace errors and warnings or not
error_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **error**, **warning**
error_events.trace.filter	Struct	Key/value pairs for event-specific filters. Valid keys are: • **error.include** (string) - like error code filters in standard plugin • **error.exclude** (string) - like error code filters in standard plugin • **warning.include** (string) - like warning code filters in standard plugin • **warning.exclude** (string) - like warning code filters in standard plugin
error_events.output.att_info	*OutputFilter*	When database or service attachment details are included.
error_events.output.items	Struct	Key/value pairs for event-specific output items. Valid keys are: • **error.codes** (bool) - Whether to fill specified field in output message. • **error.text** (bool) - Whether to fill specified field in output message. • **warning.codes** (bool) - Whether to fill specified field in output message. • **warning.text** (bool) - Whether to fill specified field in output message. **Note:** Key not present in Struct means True.

Table 13: Sweep activity events

Name	Data type	Description
sweep_events.trace.enabled	bool	Trace errors and warnings or not
sweep_events.trace.event	repeated string	If specified, emit ONLY for specified events. Valid names: **start**, **end**, **fail**, **progress**
sweep_events.output.att_info	*OutputFilter*	When database attachment details are included
sweep_events.output.perf.time_count	sint32	When total time and rows affected are included, see *Performance filter*
sweep_events.output.perf.page_io	sint32	When page I/O stats are included, see *Performance filter*
sweep_events.output.perf.row_io	sint32	When row I/O stats are included, see *Performance filter*

Table 14: Other configuration options

Name	Data type	Description
log_filename	string	Write to specified audit file (applicable only to trace audit)
output_format	*OutputFormat*	Output format for TraceEntry protocol buffer messages produced by plugin

2.3.2 TraceConfig

The *TraceConfig* is defined as follows:

```
message TraceConfig {
   GeneralFilter   trace          = 1 ;
   TopicSpec       db_events      = 2 ;
   TopicSpec       tra_events     = 3 ;
   TopicSpec       proc_events    = 4 ;
   TopicSpec       func_events    = 5 ;
   TopicSpec       trigger_events = 6 ;
   TopicSpec       ctx_events     = 7 ;
   TopicSpec       dsql_events    = 8 ;
   TopicSpec       blr_events     = 9 ;
   TopicSpec       dyn_events     = 10 ;
   TopicSpec       svc_events     = 11 ;
   TopicSpec       error_events   = 12 ;
   TopicSpec       sweep_events   = 13 ;
   string          log_filename   = 14 ;
   OutputFormat    output_format  = 15 ;
 }
```

2.3.3 GeneralFilter

The *GeneralFilter* is defined as follows:

```
message GeneralFilter {
  string database_name       = 1 ;
  string remote_process_name = 2 ;
  uint64 attachment_id       = 3 ;
  uint64 remote_pid          = 4 ;
  string remote_address      = 5 ;
  string user_name           = 6 ;
  string role_name           = 7 ;
}
```

2.3.4 TopicSpec

The *TopicSpec* message type is used to define the conditions for the emission of data about a specific type of event, and the specification of the data to be emitted.

The *TopicSpec* is defined as follows:

```
message TopicSpec {
    TopicFilter trace  = 1 ;
    TopicOutput output = 2 ;
}

message TopicFilter {
    bool                        enabled     = 1 ;
    repeated string             event       = 2 ;
```

(continues on next page)

(continued from previous page)

```
    repeated ibp.protobuf.trace.Result result      = 3 ;
    uint64                             threshold   = 4 ;
    google.protobuf.Struct             filter      = 5 ;
}

message TopicOutput {
    OutputFilter           att_info    = 1 ;
    OutputFilter           tra_info    = 2 ;
    OutputFilter           dsql_info   = 3 ;
    OutputFilter           svc_info    = 4 ;
    PerformanceFilter      perf        = 5 ;
    sint32                 inputs      = 6 ;
    google.protobuf.Struct items       = 7 ;
}

message PerformanceFilter {
    sint32  time_count  = 1 ;
    sint32  page_io     = 2 ;
    sint32  row_io      = 3 ;
}
```

2.3.5 Performance filter

Performance filter values (i.e. fields of *PerformanceFilter* message) use signed int values that indicate condition when particular performance information should be included:

Value	Description
-1	Performance information is NEVER included
0	Performance information is ALLWAYS included
n [>0]	Performance information is included WHEN action is NOT SLOWER than specified time (in *ms*)

2.3.6 Result

Result is an enumeration that defines possible event outcomes:

Name	Value	Description
RESULT_SUCCESS	0	The operation was successful
RESULT_FAILED	1	The operation failed
RESULT_UNAUTHORIZED	2	The operation failed due to an authorization failure

2.3.7 OutputFilter

OutputFilter is an enumeration that defines when detailed information about particular object should be emitted:

Name	Value	Description
OUTPUT_NEVER	0	Detail information is never emitted, the event message contains only object ID
OUTPUT_FIRST	1	Detail information is emitted ONLY on first occurent of object with particular ID, otherwise only object ID is emitted
OUTPUT_ALWAYS	2	Detail information is always emitted

2.3.8 OutputFormat

OutputFormat is an enumeration that defines output format for TraceEntry protocol buffer messages produced by plugin:

Name	Value	Description
OUTPUT_BASE64	0	Binary protobuf message encoded as BASE64 string.
OUTPUT_JSON	1	Message serialized as JSON.

2.4 Plugin output

The output of the plugin consists of protocol buffer *TraceEntry* messages, where each message contains information about one event. Messages are passed as text, either in BASE64-encoded binary format or serialized in JSON format, with each message being passed on a separate line.

2.4.1 TraceEntry

The *TraceEntry* is defined as follows:

```
message TraceEntry {
    google.protobuf.Timestamp timestamp = 1 ;

    oneof entry {
    DatabaseAttach      db_attach       = 2 ;
    DatabaseDetach      db_detach       = 3 ;
    TransactionStart    tra_start       = 4 ;
    TransactionEnd      tra_end         = 5 ;
    DSQLPrepare         dsql_prepare    = 6 ;
    DSQLStart           dsql_start      = 7 ;
    DSQLEnd             dsql_end        = 8 ;
    DSQLFree            dsql_free       = 9 ;
    ProcedureStart      proc_start      = 10 ;
    ProcedureEnd        proc_end        = 11 ;
    FunctionStart       func_start      = 12 ;
    FunctionEnd         func_end        = 13 ;
    TriggerStart        trigger_start   = 14 ;
    TriggerEnd          trigger_end     = 15 ;
    SetContext          set_context     = 16 ;
    BLRCompile          blr_compile     = 17 ;
```

(continues on next page)

(continued from previous page)

```
    BLRExecute          blr_exec        = 18 ;
    DYNExecute          dyn_exec        = 19 ;
    ServiceAttach       svc_attach      = 20 ;
    ServiceStart        svc_start       = 21 ;
    ServiceQuery        svc_query       = 22 ;
    ServiceDetach       svc_detach      = 23 ;
    Error               error           = 24 ;
    Warning             warning         = 25 ;
    SweepStart          sweep_start     = 26 ;
    SweepEnd            sweep_end       = 27 ;
    SweepFail           sweep_fail      = 28 ;
    SweepProgress       sweep_progress  = 29 ;
    }
}
```

The timestamp field *timestamp* contains the moment the event was recorded.

The oneof field *entry* is used both to identify the type of event, and at the same time to store event data as messages of a specific type (each event type has its own message type).

Oneof fields are like regular fields except all the fields in a oneof share memory, and at most one field can be set at the same time. Setting any member of the oneof automatically clears all the other members. You can check which value in a oneof is set (if any) using a special *case()* or *WhichOneof()* method, depending on your chosen language.

Note: When processing serialized messages directly in JSON format, the event type can be identified by the dictionary key used, which corresponds to the name of one of the *oneof* items.

Important: In event messages, fields marked as **OPTIONAL** may be completely missing from the message (controlled by plugin configuration).

To determine if a submessage is included, use the appropriate "has" method provided by the protocol buffer API of your choice, or check for the presence of the field name in the JSON dictionary.

2.4.2 Database attachment identification

The attachment ID is unique within a single database, but because multiple databases can be traced simultaneously, the attachment ID is not sufficient to uniquely identify database connections within a single trace session. Therefore, the plugin uses internally generated UUIDs to identify database connections.

In protobuf messages, this UUID is stored as bytes in it's binary representation. In JSON, it's serialized using BASE64 encoding.

2.4.3 DatabaseAttach

The *DatabaseAttach* message contains information about a successful or unsuccessful attempt to attach or create the database.

Table 15: DatabaseAttach fields

Name	Data type	Description
result	*Result*	The result of the attempt
create	bool	True means *create* database event, False means *attach* database event
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**

2.4.4 DatabaseDetach

The *DatabaseDetach* message contains information about a successful or unsuccessful attempt to dettach or drop the database.

Table 16: DatabaseDetach fields

Name	Data type	Description
result	*Result*	The result of the attempt
drop	bool	True means *drop* database event, False means *dettach* database event
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**

2.4.5 TransactionStart

The *TransactionStart* message contains information about an attempt to start a new transaction.

Table 17: TransactionStart fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**

2.4.6 TransactionEnd

The *TransactionEnd* message contains information about an attempt to finish a transaction.

Table 18: TransactionEnd fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
commit	bool	True for *commit* attempt, False for *rollback* attempt
retain	bool	True for context *retaining*, False for *normal* transaction end
perf	*PerformanceInfo*	Performance information. **OPTIONAL**
initial	uint64	Initial transaction ID for retained transactions
previous	uint64	Previous transaction ID for retained transactions

Important: When transactions are *retained*, it's like ending a transaction and starting a new one in a single step, and the *tra_id* contains the NEW transaction ID, not the ID of the completed transaction (which is stored in the *previous* field).

The *initial* field contains transaction ID of the first transaction in *retaining* chain.

2.4.7 ProcedureStart

The *ProcedureStart* message contains information about an attempt to start the execution of a stored procedure.

Table 19: ProcedureStart fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
name	string	Procedure name
inputs	repeated *Param*	Input parameters. **OPTIONAL**

2.4.8 ProcedureEnd

The *ProcedureEnd* message contains information about an attempt to finish the execution of a stored procedure.

Table 20: ProcedureEnd fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
name	string	Procedure name
inputs	repeated *Param*	Input parameters. **OPTIONAL**
perf	*PerformanceInfo*	Performance information. **OPTIONAL**

2.4.9 FunctionStart

The *FunctionStart* message contains information about an attempt to start the execution of a stored function.

Table 21: FunctionStart fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
name	string	Function name
inputs	repeated *Param*	Input parameters. **OPTIONAL**

2.4.10 FunctionEnd

The *FunctionEnd* message contains information about an attempt to finish the execution of a stored function.

Table 22: FunctionEnd fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
name	string	Procedure name
inputs	repeated *Param*	Input parameters. **OPTIONAL**
returns	*Param*	The return value of the function
perf	*PerformanceInfo*	Performance information. **OPTIONAL**

2.4.11 TriggerStart

The *TriggerStart* message contains information about an attempt to start the execution of a database trigger.

Table 23: TriggerStart fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
name	string	Trigger name
relation	string	Table name (empty for database triggers)
which	*TriggerWhich*	Trigger type (ALL/BEFORE/AFTER)
action	*TriggerAction*	Trigger action

2.4.12 TriggerEnd

The *TriggerEnd* message contains information about an attempt to finish the execution of a database trigger.

Table 24: TriggerEnd fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
name	string	Trigger name
relation	string	Table name (empty for database triggers)
which	*TriggerWhich*	Trigger type (ALL/BEFORE/AFTER)
action	*TriggerAction*	Trigger action
perf	*PerformanceInfo*	Performance information. **OPTIONAL**

2.4.13 SetContext

The *SetContext* message contains information about an attempt to set a new value to a context variable.

Table 25: SetContext fields

Name	Data type	Description
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
namespace	string	Context variable namespace
name	string	Context variable name
value	string	Value assigned to variable

2.4.14 DSQLPrepare

The *DSQLPrepare* message contains information about an attempt to prepare the DSQL statement.

Table 26: DSQLPrepare fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
dsql_id	uint64	DSQL statement ID
dsql_info	*DSQLInfo*	Detail information about DSQL statement. **OPTIONAL**
time	uint64	Time (ms) needed to prepare the statement

2.4.15 DSQLStart

The *DSQLStart* message contains information about an attempt to start execution of the DSQL statement.

Table 27: DSQLStart fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
dsql_id	uint64	DSQL statement ID
dsql_info	*DSQLInfo*	Detail information about DSQL statement. **OPTIONAL**
inputs	repeated *Param*	Input parameters. **OPTIONAL**

2.4.16 DSQLEnd

The *DSQLEnd* message contains information about an attempt to finish execution of the DSQL statement.

Table 28: DSQLEnd fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
dsql_id	uint64	DSQL statement ID
dsql_info	*DSQLInfo*	Detail information about DSQL statement. **OPTIONAL**
inputs	repeated *Param*	Input parameters. **OPTIONAL**
perf	*PerformanceInfo*	Performance information. **OPTIONAL**

2.4.17 DSQLFree

The *DSQLFree* message contains information about release of the DSQL statement.

Table 29: DSQLFree fields

Name	Data type	Description
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
dsql_id	uint64	DSQL statement ID
dsql_info	*DSQLInfo*	Detail information about DSQL statement. **OPTIONAL**
drop	bool	True means that the statement has been completely freed, False means that only the associated resources have been freed and the statement can be reused later.

2.4.18 BLRCompile

The *BLRCompile* message contains information about an attempt to compile the BLR statement.

Table 30: BLRCompile fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
blr_id	uint64	BLR statement ID
blr_info	*BLRInfo*	Detail information about BLR statement. **OPTIONAL**
time	uint64	Time (ms) needed to compile the statement

2.4.19 BLRExecute

The *BLRExecute* message contains information about an attempt to execute the BLR statement.

Table 31: BLRExecute fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
blr_id	uint64	BLR statement ID
blr_info	*BLRInfo*	Detail information about BLR statement. **OPTIONAL**
perf	*PerformanceInfo*	Performance information. **OPTIONAL**

2.4.20 DYNExecute

The *DYNExecute* message contains information about an attempt to execute the DYN statement.

Table 32: DYNExecute fields

Name	Data type	Description
result	*Result*	The result of the attempt
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
tra_id	uint64	Transaction ID
tra_info	*TransactionInfo*	Detail information about transaction. **OPTIONAL**
data	bytes	DYN data. **OPTIONAL**
text	string	DYN text. **OPTIONAL**

2.4.21 ServiceAttach

The *ServiceAttach* message contains information about a successful or unsuccessful attempt to attach to the service manager.

Table 33: ServiceAttach fields

Name	Data type	Description
result	*Result*	The result of the attempt
svc_id	uint64	Service attachment ID
svc_info	*ServiceInfo*	Detail information about service attachment. **OPTIONAL**

2.4.22 ServiceStart

The *ServiceStart* message contains information about a successful or unsuccessful attempt to start particular service.

Table 34: ServiceStart fields

Name	Data type	Description
result	*Result*	The result of the attempt
svc_id	uint64	Service attachment ID
svc_info	*ServiceInfo*	Detail information about service attachment. **OPTIONAL**
name	string	Service name
switches	string	Command switches passed to service

2.4.23 ServiceQuery

The *ServiceQuery* message contains information about an attempt to communicate with a running service.

Table 35: ServiceQuery fields

Name	Data type	Description
result	*Result*	The result of the attempt
svc_id	uint64	Service attachment ID
svc_info	*ServiceInfo*	Detail information about service attachment. **OPTIONAL**
name	string	Service name
sent	bytes	Data sent by service
received	bytes	Data received by service

2.4.24 ServiceDetach

The *ServiceDetach* message contains information about a successful or unsuccessful attempt to dettach from the service manager.

Table 36: ServiceDetach fields

Name	Data type	Description
result	*Result*	The result of the attempt
svc_id	uint64	Service attachment ID
svc_info	*ServiceInfo*	Detail information about service attachment. **OPTIONAL**

2.4.25 Error

The *Error* message contains information about the occurrence of an error in the database connection or service.

Note: This message type is used for both, database and service errors. It uses oneof fields to identify the source of error. The *id* oneof is always present, but *info* oneof could be missing as contains only **OPTIONAL** fields.

Table 37: Error fields

Name	Data type	Description
att_uuid	bytes	Database attachment ID. Part of *id* oneof.
svc_id	uint64	Service attachment ID. Part of *id* oneof.
att_info	*AttachmentInfo*	Detail information about database attachment. Part of *info* oneof. **OPTIONAL**
svc_info	*ServiceInfo*	Detail information about service attachment. Part of *info* oneof. **OPTIONAL**
codes	repeated uint64	Error codes
text	string	Error text

2.4.26 Warning

The *Warning* message contains information about the occurrence of a warning in the database connection or service.

Note: This message type is used for both, database and service warning. It uses oneof fields to identify the source of warning. The *id* oneof is always present, but *info* oneof could be missing as contains only **OPTIONAL** fields.

Table 38: Warning fields

Name	Data type	Description
att_uuid	bytes	Database attachment ID. Part of *id* oneof.
svc_id	uint64	Service attachment ID. Part of *id* oneof.
att_info	*AttachmentInfo*	Detail information about database attachment. Part of *info* oneof. **OPTIONAL**
svc_info	*ServiceInfo*	Detail information about service attachment. Part of *info* oneof. **OPTIONAL**
codes	repeated uint64	Warning codes
text	string	Warning text

2.4.27 SweepStart

The *SweepStart* message contains information about start of sweep process.

Table 39: SweepStart fields

Name	Data type	Description
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
oit	uint64	OIT value
ost	uint64	OST value
oat	uint64	OAT value
next	uint64	Next transaction ID

2.4.28 SweepEnd

The *SweepEnd* message contains information about end of sweep process.

Table 40: SweepEnd fields

Name	Data type	Description
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
oit	uint64	OIT value
ost	uint64	OST value
oat	uint64	OAT value
next	uint64	Next transaction ID
perf	*PerformanceInfo*	Summary performance information. **OPTIONAL**

2.4.29 SweepFail

The *SweepFail* message contains information about failure of sweep process.

Table 41: SweepFail fields

Name	Data type	Description
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**

2.4.30 SweepProgress

The *SweepProgress* message contains information about progress of sweep process (one table was swept).

Table 42: SweepProgress fields

Name	Data type	Description
att_uuid	bytes	Database attachment ID
att_info	*AttachmentInfo*	Detail information about attachment. **OPTIONAL**
perf	*PerformanceInfo*	New performance information. **OPTIONAL**

2.4.31 TriggerWhich

TriggerWhich is an enumeration that defines when trigger is fired:

Name	Value	Description
WHICH_ALL	0	Database trigger.
WHICH_BEFORE	1	Trigger fired before table row change.
WHICH_AFTER	2	Trigger fired after table row change.

2.4.32 TriggerAction

TriggerAction is an enumeration that defines the action in the context of which the trigger is fired:

Name	Value	Description
TRIGGER_UNKNOWN	0	Unknown action.
TRIGGER_INSERT	1	INSERT row.
TRIGGER_UPDATE	2	UPDATE row.
TRIGGER_DELETE	3	DELETE row.
TRIGGER_CONNECT	4	Attach database.
TRIGGER_DISCONNECT	5	Dettach database.
TRIGGER_TRANS_START	6	Transaction start.
TRIGGER_TRANS_COMMIT	7	Transaction COMMIT.
TRIGGER_TRANS_ROLLBACK	8	Transaction ROLLBACK.
TRIGGER_DDL	9	DDL operation.

2.4.33 PerformanceInfo

The *PerformanceInfo* message contains performance-related information associated with particular event.

Table 43: PerformanceInfo fields

Name	Data type	Description
time	uint64	Total operation time in milliseconds
records	uint64	Records fetched/processed
fetches	uint64	Number of data page fetches from cache
reads	uint64	Number of data page reads from disk to cache
marks	uint64	Number of data page changes in cache
writes	uint64	Number of data page writes from cache to disk
tables	repeated *TableCounters*	Row I/O stats for processed tables

2.4.34 TableCounters

The *TableCounters* message contains performance-related information for single table.

Table 44: TableCounters fields

Name	Data type	Description
name	string	Table name
seq_reads	uint64	Number of rows read sequentially
idx_reads	uint64	Number of rows read via index
updates	uint64	Number of update rows
inserts	uint64	Number of inserted rows
deletes	uint64	Number of deleted rows
backouts	uint64	Number of removed latest (uncommitted) record versions
purges	uint64	Number of removed intermediate old record versions (outdated versions found while chasing)
expunges	uint64	Number of whole version chains removed, along with the primary version (Record is deleted and nobody is interested)
locks	uint64	Number of rows selected using WITH LOCK clause
waits	uint64	Number of attempts to update/delete/lock row owned by a concurrent active transaction
conflicts	uint64	Number of unsuccessful attempts to update/delete/lock row owned by a concurrent active transaction (UPDATE CONFLICT is reported)
back_reads	uint64	Number of record versions chased while finding a visible one
fragment_reads	uint64	Number of fragmented rows read (means extra page fetches/reads)
rpt_reads	uint64	Number of repeated reads of particular row

2.4.35 Param

The *Param* message contains information about single parameter associated with particular event.

Table 45: Param fields

Name	Data type	Description
type	string	Value type
value	string	Parameter value

2.4.36 AttachmentInfo

The *AttachmentInfo* message contains detail information about database attachment.

Table 46: AttachmentInfo fields

Name	Data type	Description
att_id	uint64	Database attachment ID (from Firebird)
database_name	string	Database name
user_name	string	User name
role_name	string	Role used for attachment
charset	string	Connection character set (used for data transfer)
remote_protocol	string	Protocol used for remote access to Firebird
remote_address	string	Client address
remote_pid	int64	Client PID
remote_process_name	string	Client process name

2.4.37 TransactionInfo

The *TransactionInfo* message contains detail information about transaction.

Table 47: TransactionInfo fields

Name	Data type	Description
read_only	bool	Whether transaction is read only
wait	sint32	Transaction timeout parameter
isolation	uint32	Isolation level (code used by Firebird)

Table 48: Firebird isolation level codes

Number	Isolation level
1	CONSISTENCY
2	CONCURRENCY
3	READ COMMITTED | REC_VERSION
4	READ COMMITTED | NO_REC_VERSION
5	READ COMMITTED | READ_CONSISTENCY

2.4.38 DSQLInfo

The *DSQLInfo* message contains detail information about DSQL statement.

Table 49: DSQLInfo fields

Name	Data type	Description
text	string	SQL command
plan	string	Execution plan in standard (legacy) format
explained	string	Execution plan in in (explained) format

2.4.39 BLRInfo

The *BLRInfo* message contains detail information about BLR statement.

Table 50: BLRInfo fields

Name	Data type	Description
data	bytes	Binary BLR representation
text	string	Text BLR representation

2.4.40 ServiceInfo

The *ServiceInfo* message contains detail information about service attachment.

Table 51: ServiceInfo fields

Name	Data type	Description
user_name	string	User name
role_name	string	Role used for attachment
charset	string	Connection character set (used for data transfer)
remote_protocol	string	Protocol used for remote access to Firebird
remote_address	string	Client address
remote_pid	int64	Client PID
remote_process_name	string	Client process name
pid	int64	Firebird process PID?

CHAPTER

THREE

UTILITIES

The Trace plugin comes with several tools written in Python. These tools are:

- *trace-conf* - GUI Editor for Trace Plugin Configuration
- *trace-debug* - Tool for fast tracing with output of events in the selected format to the screen or to a file.
- *rt-mon* - A simple real-time connection and transaction monitor serving as a sample application for using the plugin.

3.1 Installation

All tools require Python 3.8 or later. Installation package with tools is stored in *python* subdirectory of the distrubution package, and can be installed with standard *pip* utility:

```
pip install ibp_trace-1.0.0-py3-none-any.whl
```

This will install all required dependencies (like firebird-driver for Python) and place all tools on the search path.

Note: The distribution package contains a compiled version of all tools for the Windows platform (7 or later). This version can be used directly and does not require Python to be installed.

Important: If you want to install Python on Windows, don't use Python from the Windows Store! This version has compatibility issues and the tools may not work properly. Always use the installer packages from python.org.

3.2 Trace Plugin Configuration Editor

```
usage: trace-conf [-h] [--version] [config]

IBPhoenix trace plugin configuration editor.

positional arguments:
  config      Configuration file (default: None)

optional arguments:
  -h, --help  show this help message and exit
  --version   show program's version number and exit
```

The GUI editor provides next interface.

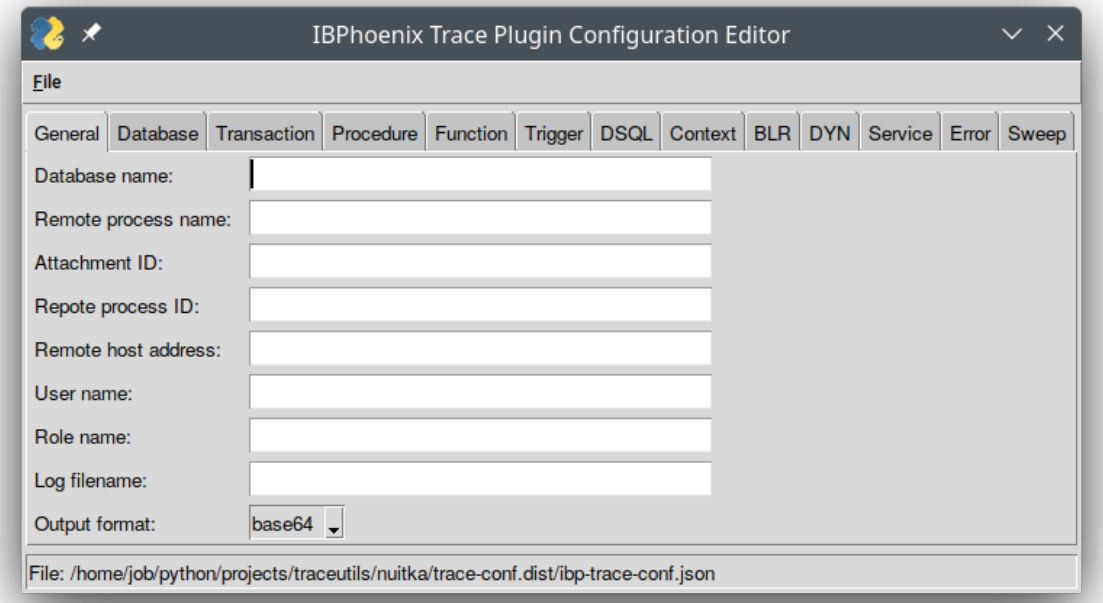


Fig. 1: The first tab is for setting of *general filters*.

Important: You MUST fill in at least one condition option, or plugin will not produce any output. If more options are set, the logic operation between them is AND.

Table 1: General tab description

Name	Description
Database name	regex for database name
Remote process name	regex for remote process name
Attachment ID	Trace single attachment with specific ID
Remote process ID	Trace all attachments from single remote process specified by PID
Remote host address	Trace all attachments/processes from single host
User name	Trace all attachments from specified user
Role name	Trace all attachments for specific role
Log filename	Write to specified audit file (applicable only to trace audit)
Output format	Trace plugin output format

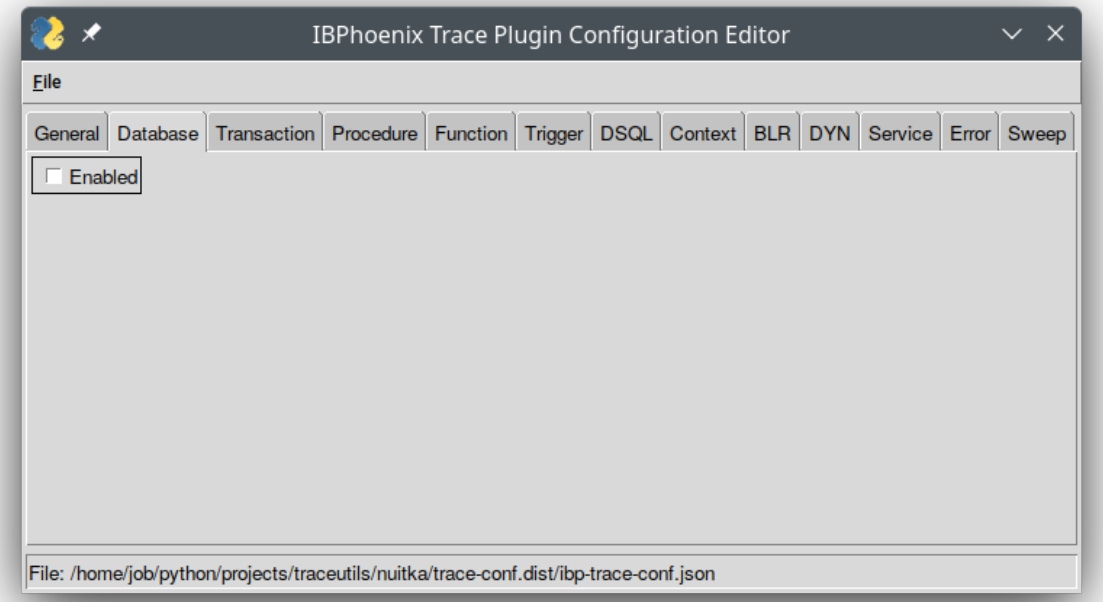


Fig. 2: Options in all other tabs are disabled by default.
Options are shown only when trace for particular category is enabled.

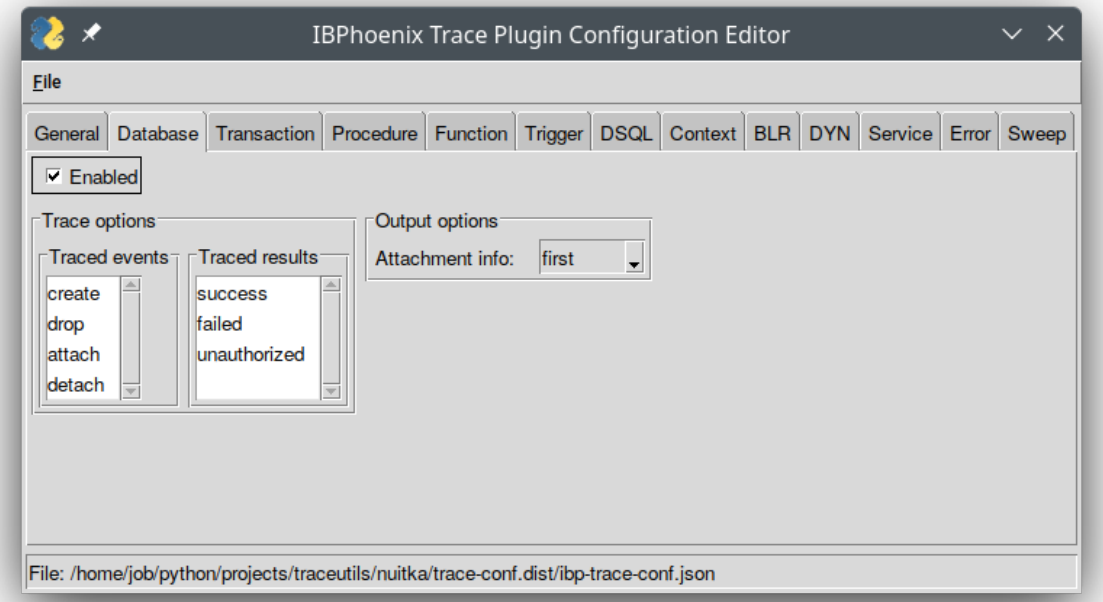


Fig. 3: The second tab is for setting of *database events*.

Table 2: Database tab description

Name	Description
Enabled	Trace database attachment events or not
Trace events	If specified, emit ONLY for specified events. You may select multiple options in list.
Trace results	If specified, emit only for specified results. You may select multiple options in list.
Attachment info	When database attachment details are included

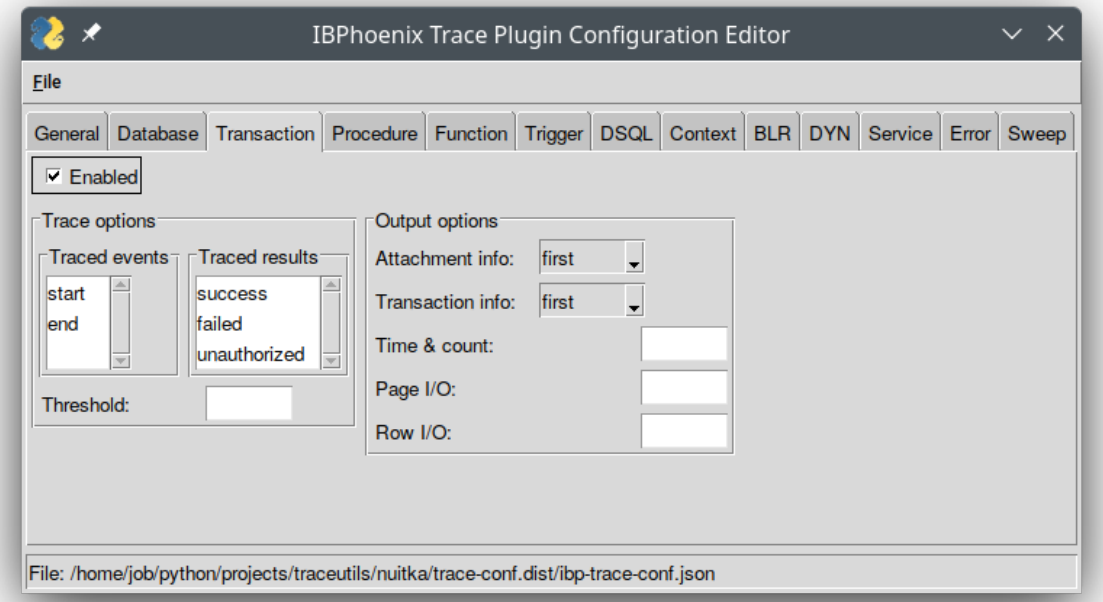


Fig. 4: The third tab is for setting of *transaction events*.

Table 3: Transaction tab description

Name	Description
Enabled	Trace transaction events or not
Trace events	If specified, emit ONLY for specified events. You may select multiple options in list.
Trace results	If specified, emit only for specified results. You may select multiple options in list.
Threshold	If specified, emit only when action is slower than threshold (ms)
Attachment info	When database attachment details are included
Transaction info	When transaction details are included
Time & count	When total time and rows affected are included, see *Performance filter*.
Page I/O	When page I/O stats are included, see *Performance filter*.
Row I/O	When row I/O stats are included, see *Performance filter*.

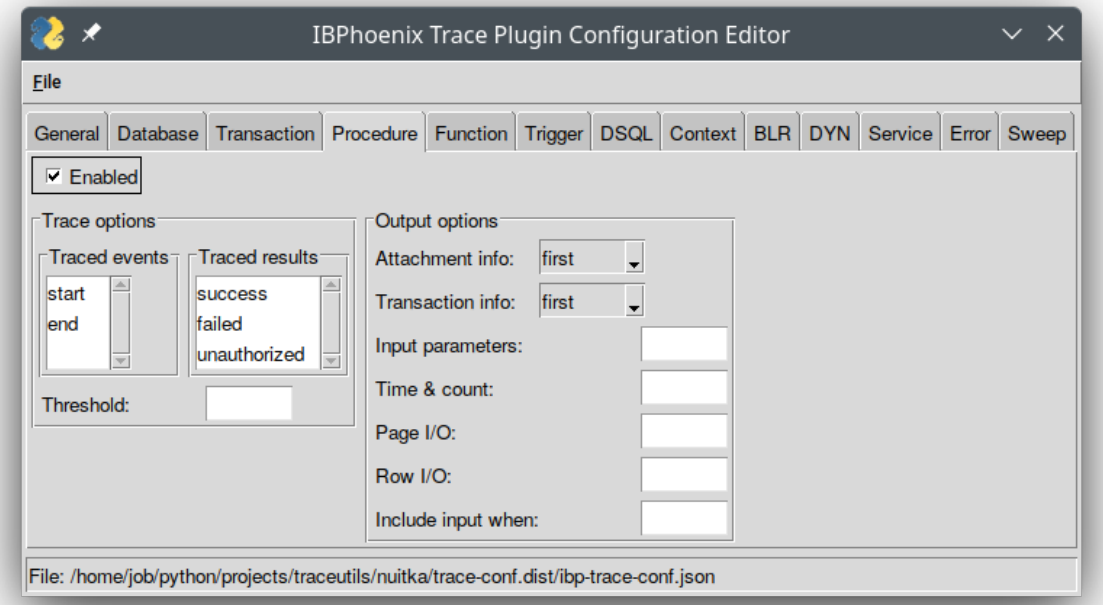


Fig. 5: The fourth tab is for setting of *stored procedure events*.

Table 4: Procedure tab description

Name	Description
Enabled	Trace procedure events or not
Trace events	If specified, emit ONLY for specified events. You may select multiple options in list.
Trace results	If specified, emit only for specified results. You may select multiple options in list.
Threshold	If specified, emit only when action is slower than threshold (ms)
Attachment info	When database attachment details are included
Transaction info	When transaction details are included
Input parameters	Include input parameters: -1 = all, 0 = none, *n* = up to <n> parameters
Time & count	When total time and rows affected are included, see *Performance filter*.
Page I/O	When page I/O stats are included, see *Performance filter*.
Row I/O	When row I/O stats are included, see *Performance filter*.
Include input when	WHEN include input parameters (if Input parameters != 0): • -1 = if both start & end are traced, then only for start; if only start or end is traced, then it's like always • 0 = allways (for both start and end events) • >0 = only for end event when execution is slower than value (ms)

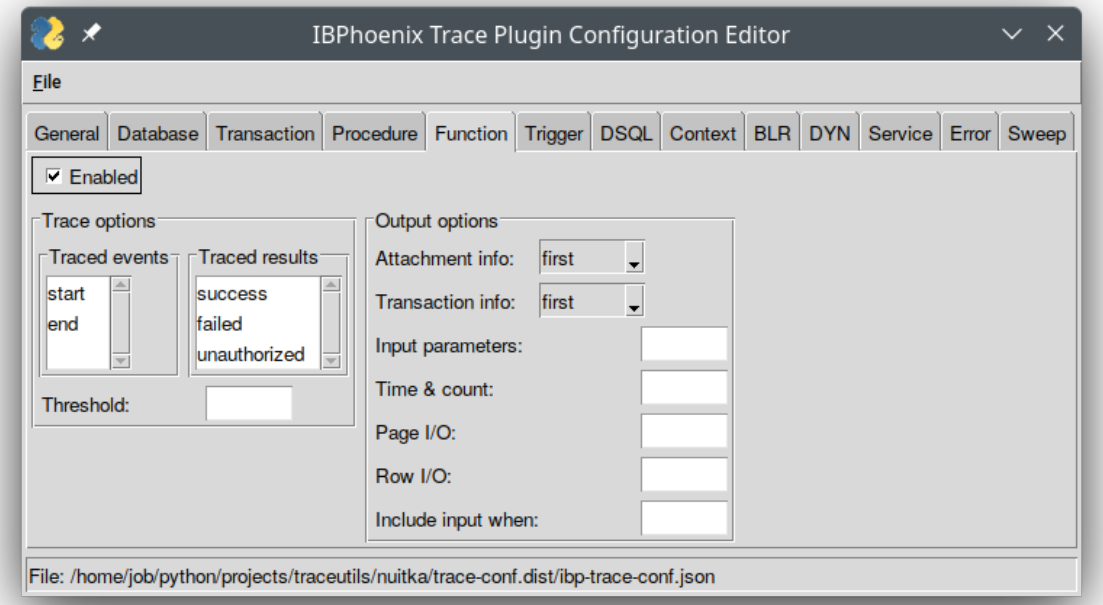


Fig. 6: The fifth tab is for setting of *stored function events*.

Table 5: Function tab description

Name	Description
Enabled	Trace function events or not
Trace events	If specified, emit ONLY for specified events. You may select multiple options in list.
Trace results	If specified, emit only for specified results. You may select multiple options in list.
Threshold	If specified, emit only when action is slower than threshold (ms)
Attachment info	When database attachment details are included
Transaction info	When transaction details are included
Input parameters	Include input parameters: -1 = all, 0 = none, *n* = up to <n> parameters
Time & count	When total time and rows affected are included, see *Performance filter*.
Page I/O	When page I/O stats are included, see *Performance filter*.
Row I/O	When row I/O stats are included, see *Performance filter*.
Include input when	WHEN include input parameters (if Input parameters != 0): • -1 = if both start & end are traced, then only for start; if only start or end is traced, then it's like always • 0 = allways (for both start and end events) • >0 = only for end event when execution is slower than value (ms)

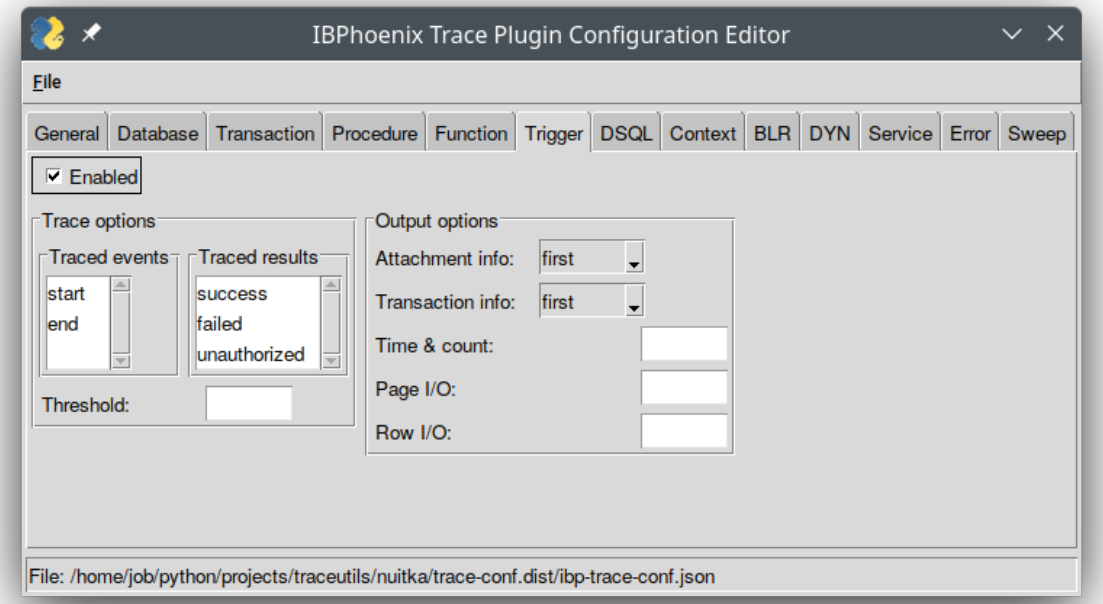


Fig. 7: The sixth tab is for setting of *trigger events*.

Table 6: Trigger tab description

Name	Description
Enabled	Trace trigger events or not
Trace events	If specified, emit ONLY for specified events. You may select multiple options in list.
Trace results	If specified, emit only for specified results. You may select multiple options in list.
Threshold	If specified, emit only when action is slower than threshold (ms)
Attachment info	When database attachment details are included
Transaction info	When transaction details are included
Time & count	When total time and rows affected are included, see *Performance filter*.
Page I/O	When page I/O stats are included, see *Performance filter*.
Row I/O	When row I/O stats are included, see *Performance filter*.

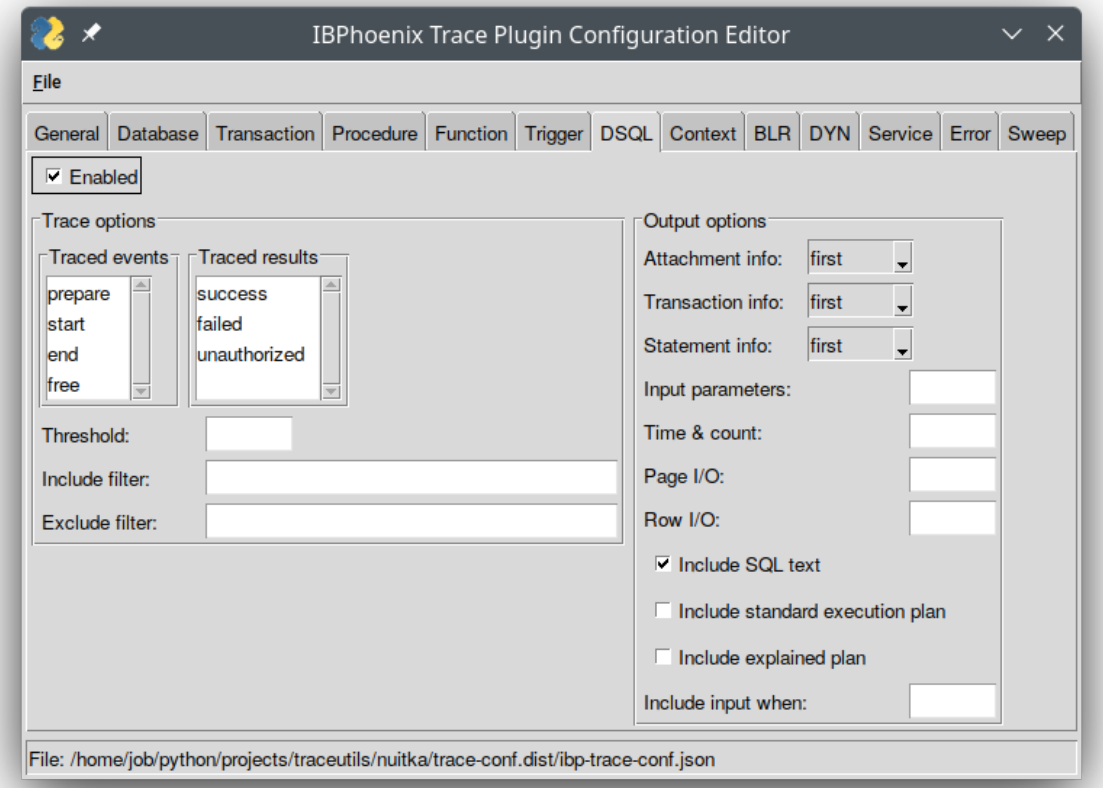


Fig. 8: The seventh tab is for setting of *DSQL statement events*.

Table 7: DSQL tab description

Name	Description
Enabled	Trace DSQL statement events or not
Trace events	If specified, emit ONLY for specified events. You may select multiple options in list.
Trace results	If specified, emit only for specified results. You may select multiple options in list.
Threshold	If specified, emit only when action is slower than threshold (ms)
Include filter	Emit ONLY when SQL contains text
Exclude filter	DO NOT emit when SQL contains text
Attachment info	When database attachment details are included
Transaction info	When transaction details are included
Statement info	When DSQL statement details are included
Input parameters	Include input parameters: -1 = all, 0 = none, *n* = up to <n> parameters
Time & count	When total time and rows affected are included, see *Performance filter*.
Page I/O	When page I/O stats are included, see *Performance filter*.
Row I/O	When row I/O stats are included, see *Performance filter*.
Include SQL text	Self-explanatory
Include standard execution plan	Self-explanatory
Include explained plan	Self-explanatory
Include input when	WHEN include input parameters (if Input parameters != 0): • -1 = if both start & end are traced, then only for start; if only start or end is traced, then it's like always • 0 = allways (for both start and end events) • >0 = only for end event when execution is slower than value (ms)

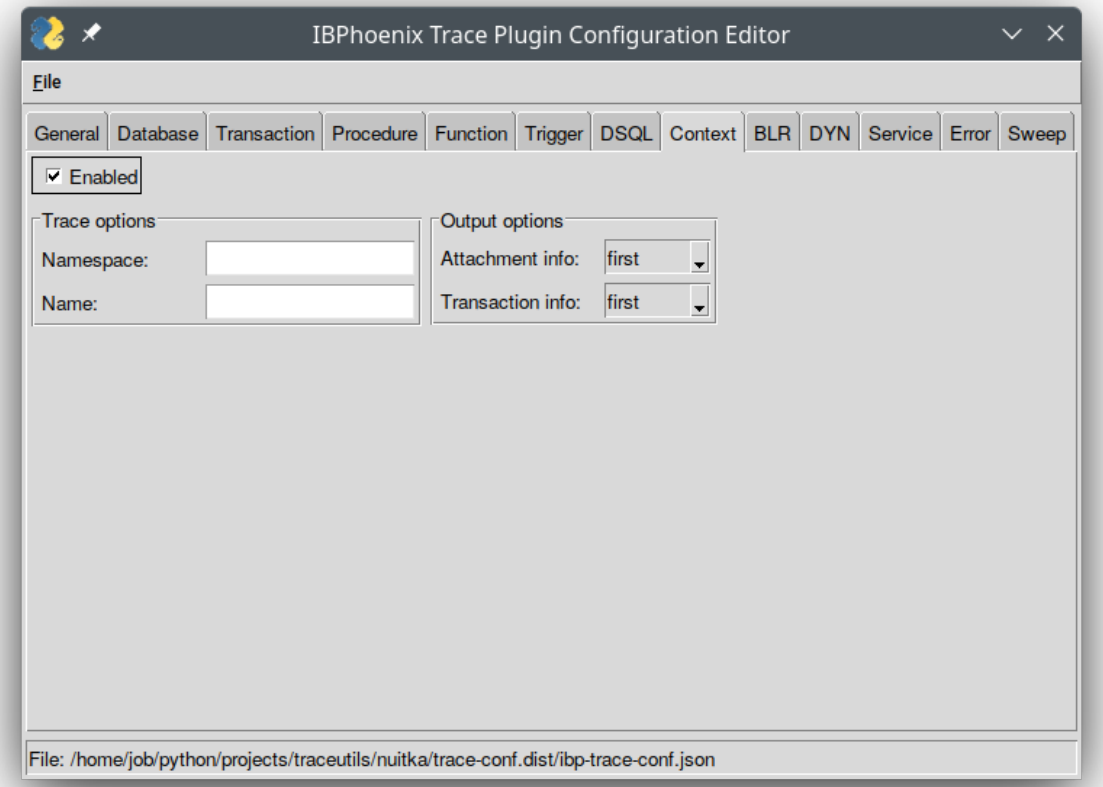


Fig. 9: The eighth tab is for setting of *context variable events*.

Table 8: Context tab description

Name	Description
Enabled	Trace trigger events or not
Namepace	Emit only for specified namespace.
Name	Emit only for specified variable name.
Attachment info	When database attachment details are included
Transaction info	When transaction details are included

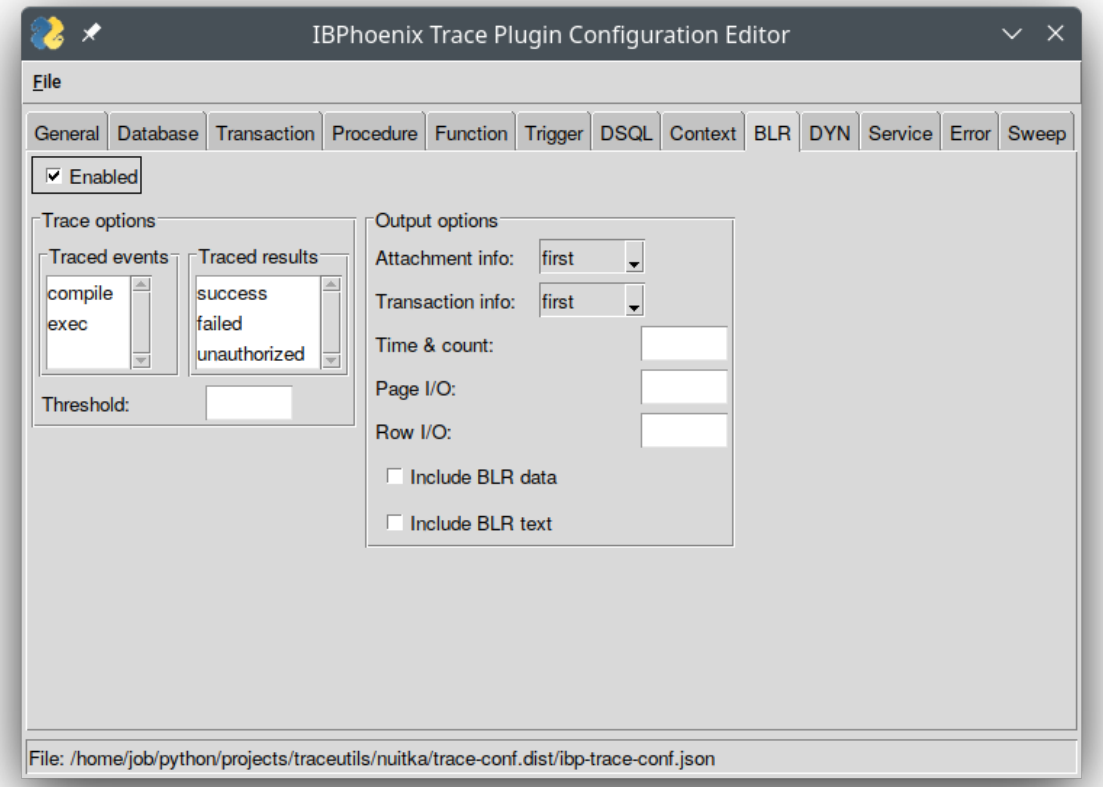


Fig. 10: The ninth tab is for setting of *BLR statement events*.

Table 9: BLR tab description

Name	Description
Enabled	Trace BLR statement events or not
Trace events	If specified, emit ONLY for specified events. You may select multiple options in list.
Trace results	If specified, emit only for specified results. You may select multiple options in list.
Threshold	If specified, emit only when action is slower than threshold (ms)
Attachment info	When database attachment details are included
Transaction info	When transaction details are included
Time & count	When total time and rows affected are included, see *Performance filter*.
Page I/O	When page I/O stats are included, see *Performance filter*.
Row I/O	When row I/O stats are included, see *Performance filter*.
Include BLR data	Self-explanatory
Include BLR text	Self-explanatory

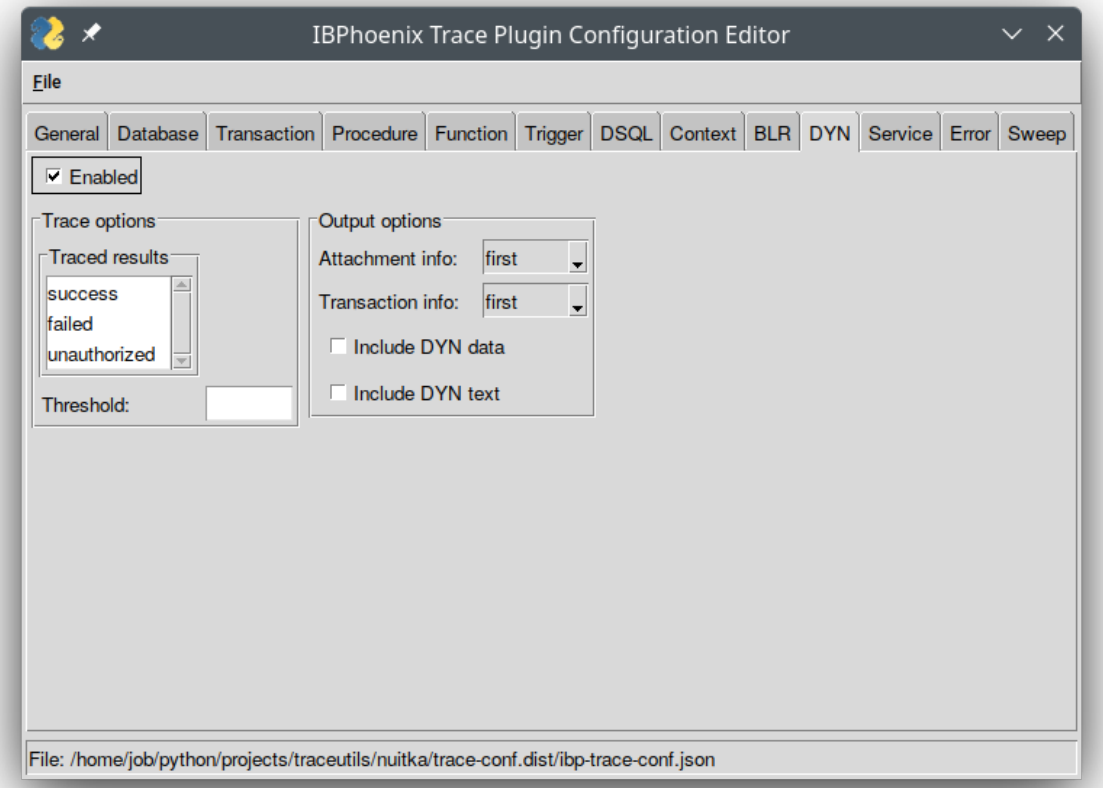


Fig. 11: The tenth tab is for setting of *DYN statement events*.

Table 10: DYN tab description

Name	Description
Enabled	Trace DYN statement events or not
Trace events	If specified, emit ONLY for specified events. You may select multiple options in list.
Trace results	If specified, emit only for specified results. You may select multiple options in list.
Threshold	If specified, emit only when action is slower than threshold (ms)
Attachment info	When database attachment details are included
Transaction info	When transaction details are included
Include DYN data	Self-explanatory
Include DYN text	Self-explanatory

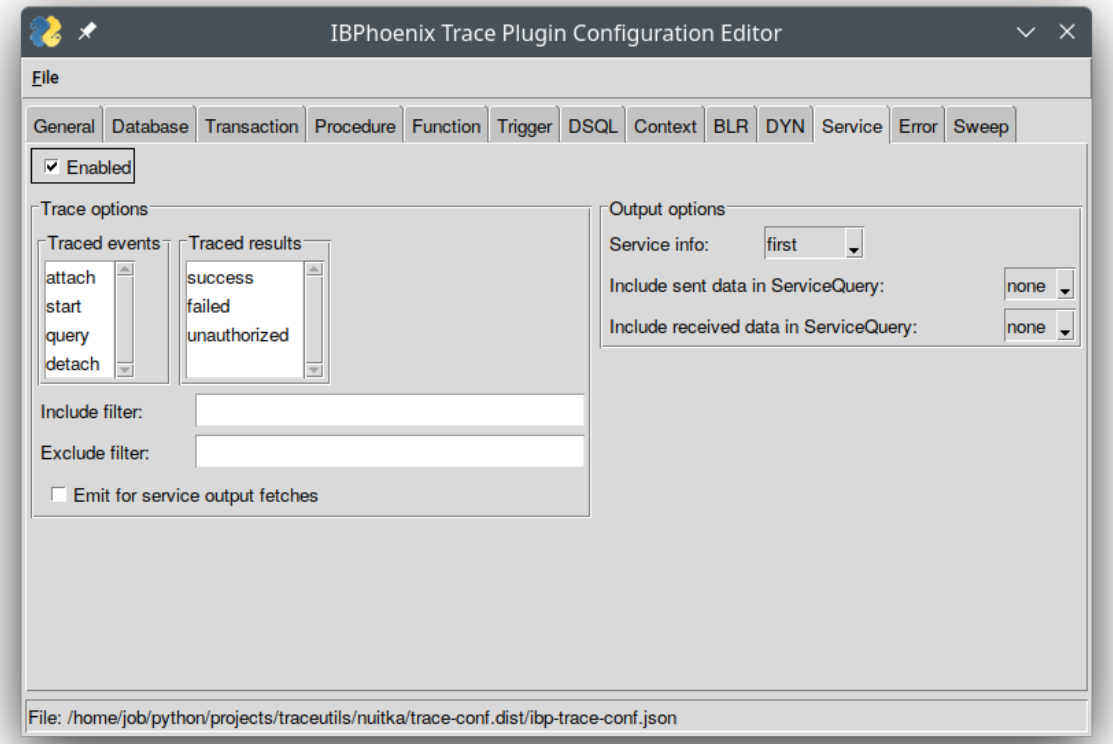


Fig. 12: The eleventh tab is for setting of *service events*.

Table 11: Service tab description

Name	Description
Enabled	Trace service events or not
Trace events	If specified, emit ONLY for specified events. You may select multiple options in list.
Trace results	If specified, emit only for specified results. You may select multiple options in list.
Emit for service output fetches	When False, does not emit TraceEntry when *recv_items* contain *isc_info_svc_line* or *isc_info_svc_to_eof*
Include filter	Emit ONLY when service name matches given regex
Exclude filter	DO NOT emit when service name matches given regex
Service info	When service attachment details are included
Include sent data in ServiceQuery	See description in *service events* (svc_events.output.items)
Include received data in ServiceQuery	See description in *service events* (svc_events.output.items)

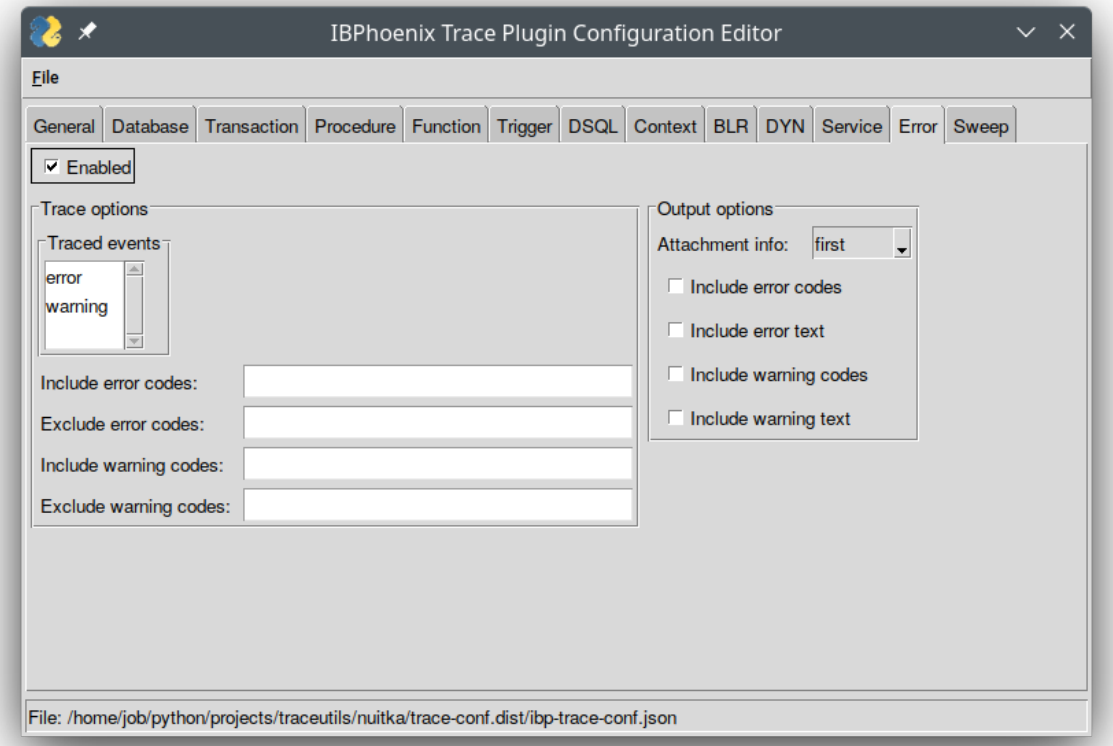


Fig. 13: The twelfth tab is for setting of *error and warning events*.

Table 12: Error tab description

Name	Description
Enabled	Trace error and warning events or not
Trace events	If specified, emit ONLY for specified events. You may select multiple options in list.
Include error codes	Like error code filters in standard plugin
Exclude error codes	Like error code filters in standard plugin
Include warning codes	Like warning code filters in standard plugin
Exclude warning codes	Like warning code filters in standard plugin
Attachment info	When database or service attachment details are included
Include error codes	Self-explanatory
Include error text	Self-explanatory
Include warning codes	Self-explanatory
Include warning text	Self-explanatory

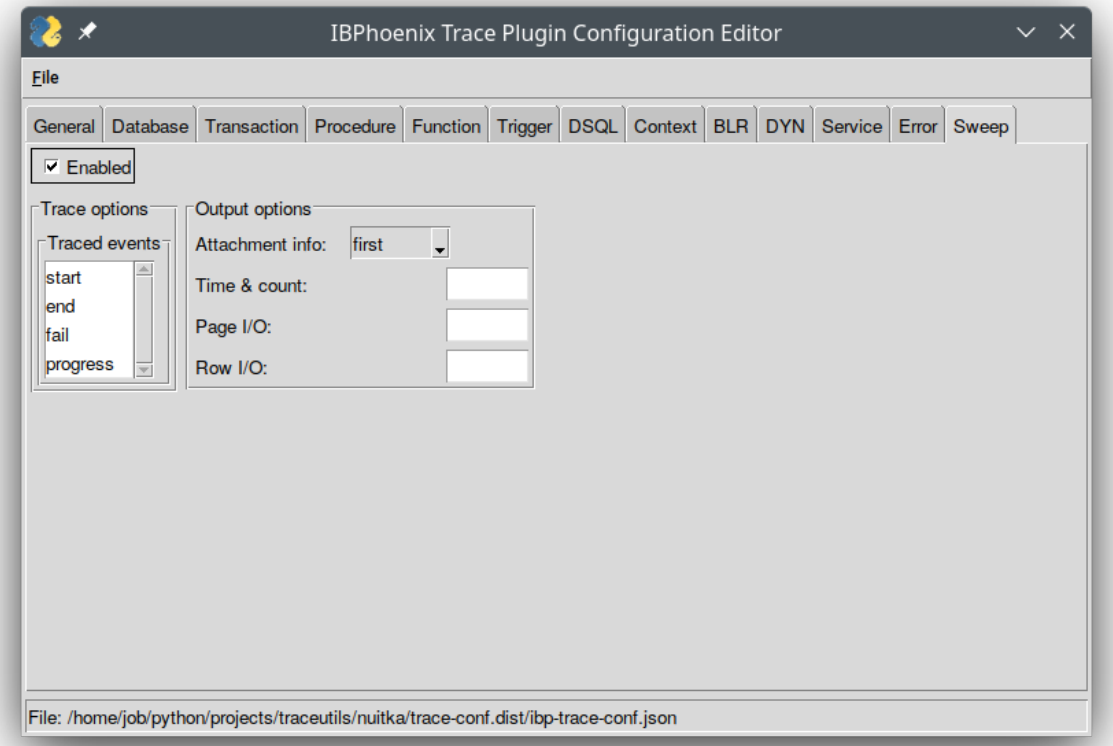


Fig. 14: The thirteenth tab is for setting of *sweep events*.

Table 13: Sweep tab description

Name	Description
Enabled	Trace sweep events or not
Trace events	If specified, emit ONLY for specified events. You may select multiple options in list.
Attachment info	When database attachment details are included
Time & count	When total time and rows affected are included, see *Performance filter*.
Page I/O	When page I/O stats are included, see *Performance filter*.
Row I/O	When row I/O stats are included, see *Performance filter*.

3.3 Trace session test and recording utility

This utility runs user trace session with specified configuration, and prints trace plugin output.

Usage:

```
trace-debug [OPTIONS] CONFIG
```

Table 14: Arguments

Name	Value	Description
config	PATH	Trace plugin configuration file [default: None][required]

Table 15: Options

Name	Value	Description
–server, -s	TEXT	Firebird server [default: localhost]
–user, -u	TEXT	Firebird user name [default: SYSDBA]
–password, -p	TEXT	User password [default: masterkey]
–format, -f	[raw|text|json|base64]	Output format [default: raw]
–output, -o	FILENAME	Output file [default: None]
–install-completion	[bash|zsh|fish|powershell|pwsh]	Install completion for the specified shell. [default: None]
–show-completion	[bash|zsh|fish|powershell|pwsh]	Show completion for the specified shell, to copy it or customize the installation. [default: None]
–help		Show help message and exit.

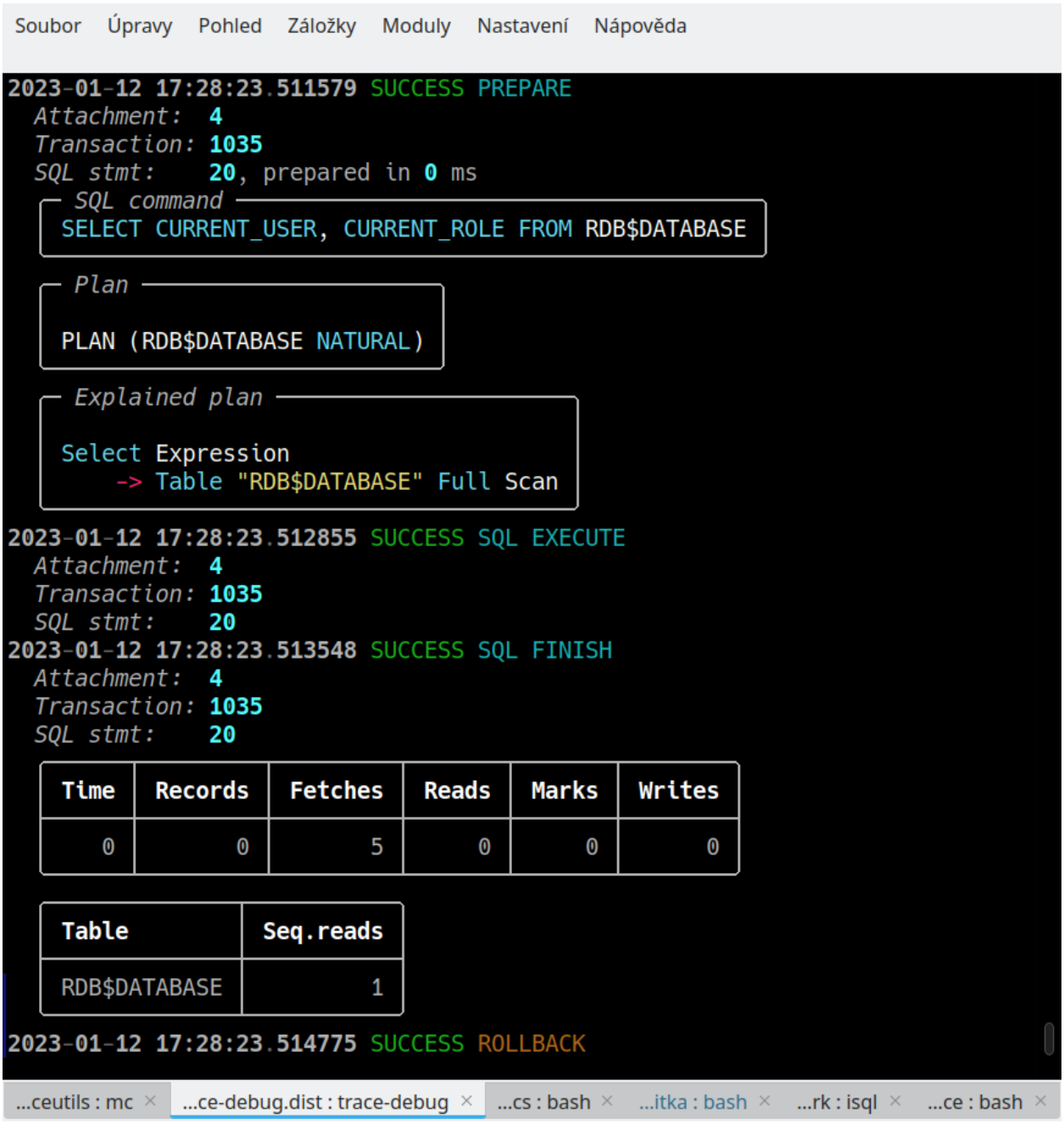
Default **"raw"** format prints plugin output "as is", i.e. JSON or BASE64 according to plugin configuration. To enforce JSON or BASE64 format, use appropriate *–format* option. The **"text"** format prints nicely formatted trace messages.

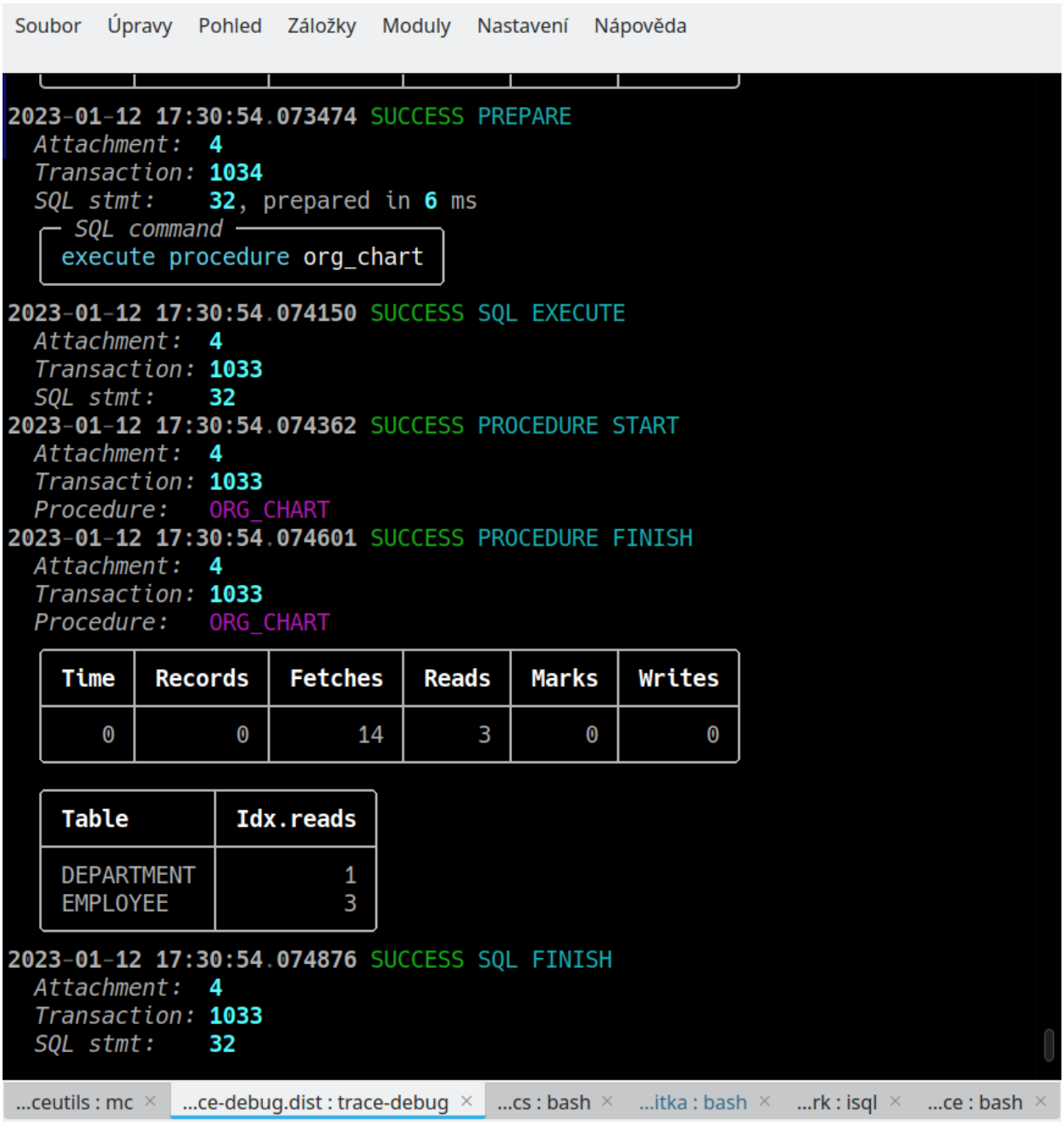
Important: Use *Q*, *q* or *ESC* keys to end execution of the debugger. Terminating via ^*C* will NOT end the trace session properly!

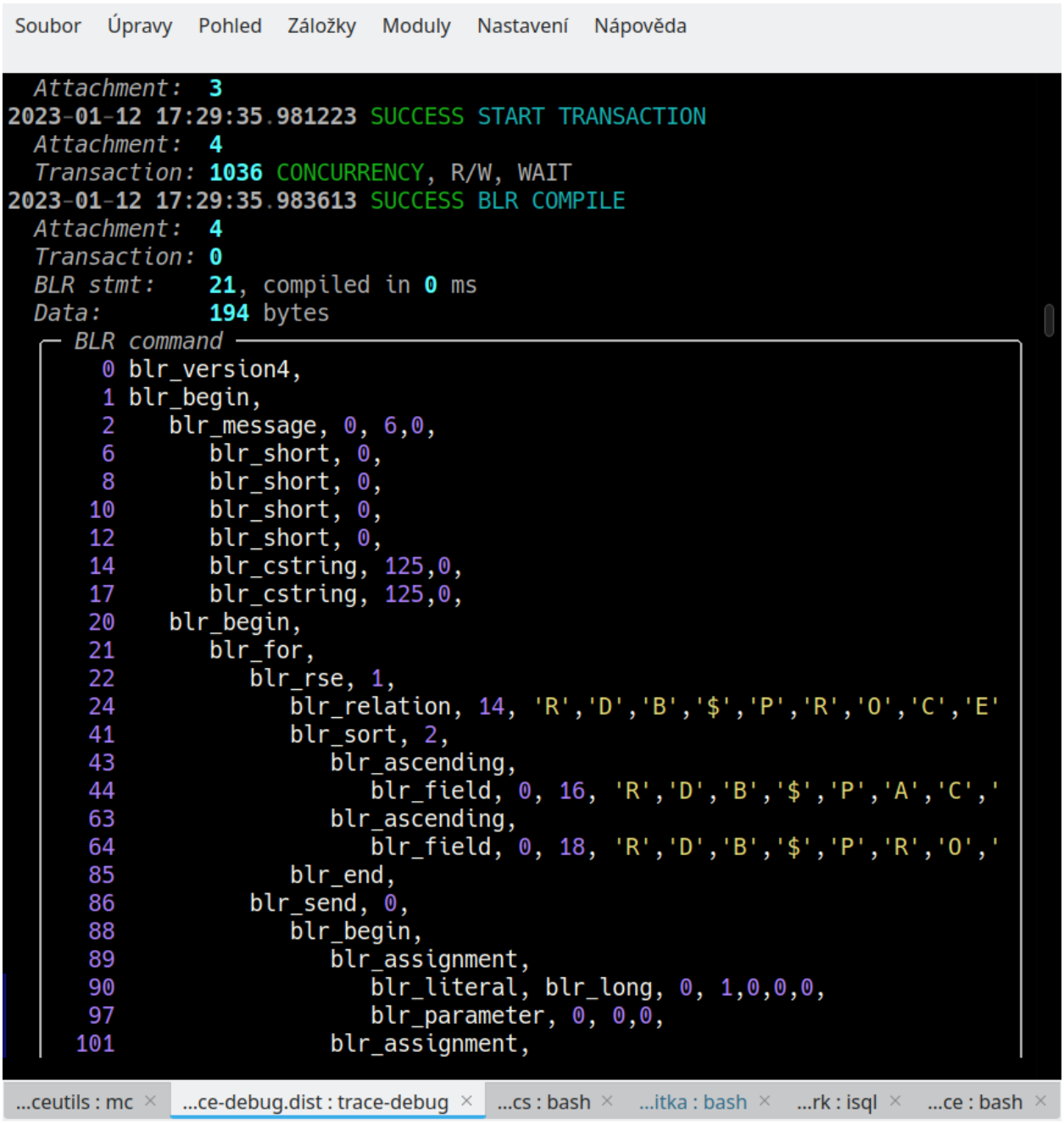
By default, debugger prints output on terminal (uses colors if possible). To save output to file, use ONLY **–output** option, as stdout redirection is not supported.

When output is stored in file using *–output* options, the debugger displays a progress information about lines written so far.

Example of text output:


Soubor Úpravy Pohled Záložky Moduly Nastavení Nápověda
2023-01-12 17:28:23.511579 SUCCESS PREPARE
Attachment: 4
Transaction: 1035
SQL stmt: 20, prepared in 0 ms
SQL command
SELECT CURRENT_USER, CURRENT_ROLE FROM RDB$DATABASE
Plan
PLAN (RDB$DATABASE NATURAL)
Explained plan
Select Expression
-> Table "RDB$DATABASE" Full Scan
2023-01-12 17:28:23.512855 SUCCESS SQL EXECUTE
Attachment: 4
Transaction: 1035
SQL stmt: 20
2023-01-12 17:28:23.513548 SUCCESS SQL FINISH
Attachment: 4
Transaction: 1035
SQL stmt: 20
Time Records Fetches Reads Marks Writes
0 0 5 0 0 0
Table Seq.reads
RDB$DATABASE 1
2023-01-12 17:28:23.514775 SUCCESS ROLLBACK
...ceutils : mc
...ce-debug.dist : trace-debug
...cs : bash
...itka : bash
...rk : isql
...ce : bash


Soubor Úpravy Pohled Záložky Moduly Nastavení Nápověda
2023-01-12 17:30:54.073474 SUCCESS PREPARE
Attachment: 4
Transaction: 1034
SQL stmt: 32, prepared in 6 ms
SQL command
execute procedure org_chart
2023-01-12 17:30:54.074150 SUCCESS SQL EXECUTE
Attachment: 4
Transaction: 1033
SQL stmt: 32
2023-01-12 17:30:54.074362 SUCCESS PROCEDURE START
Attachment: 4
Transaction: 1033
Procedure: ORG_CHART
2023-01-12 17:30:54.074601 SUCCESS PROCEDURE FINISH
Attachment: 4
Transaction: 1033
Procedure: ORG_CHART
Time Records Fetches Reads Marks Writes
0 0 14 3 0 0
Table Idx.reads
DEPARTMENT 1
EMPLOYEE 3
2023-01-12 17:30:54.074876 SUCCESS SQL FINISH
Attachment: 4
Transaction: 1033
SQL stmt: 32
...ceutils : mc
...ce-debug.dist : trace-debug
...cs : bash
...itka : bash
...rk : isql
...ce : bash



3.4 Real-time connection and transaction monitor

This sample application uses the IBPhoenix trace plugin to collect attachment and transaction information for a particular database and displays it in real time.

With compact data, it can handle more events per second than any application that uses a standard trace plugin. However, it still has its scalability limits.

Usage:

```
rt-mon [OPTIONS] DATABASE
```

Table 16: Arguments

Name	Value	Description
database	TEXT	Database [default: None] [required]

Table 17: Options

Name	Value	Description
–user, -u	TEXT	Firebird user name [default: SYSDBA]
–password, -p	TEXT	User password [default: masterkey]
–server, -s	TEXT	Firebird server [default: localhost]
–install-completion	[bash|zsh|fish|powershell|pwsh]	Install completion for the specified shell. [default: None]
–show-completion	[bash|zsh|fish|powershell|pwsh]	Show completion for the specified shell, to copy it or customize the installation. [default: None]
–help		Show help message and exit.

Utility output: