The Firebird Butler

An Introduction

Pavel Císař

Saturnin/Saturnin SDK - lead developer
IBPhoenix & Firebird Project

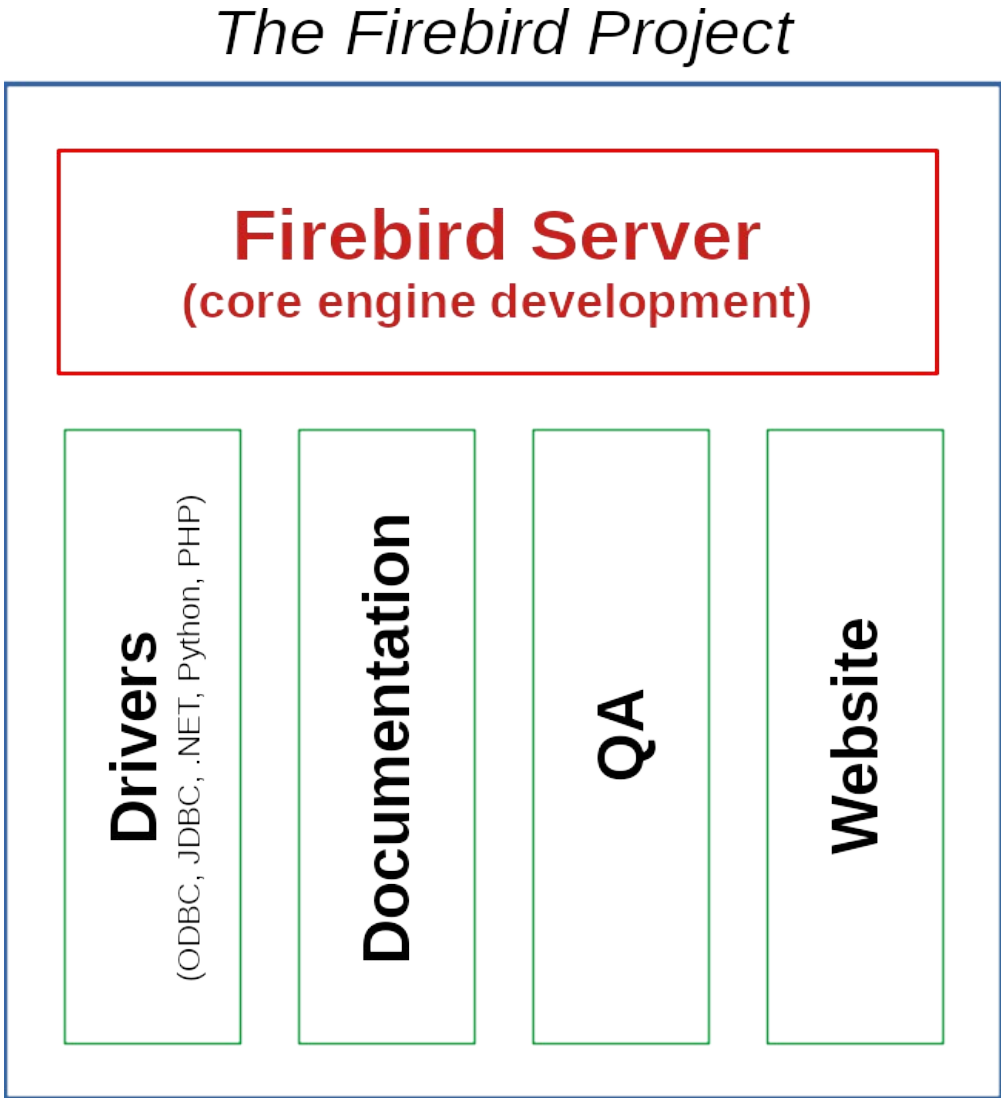
Firebird Conference 2019, Berlin

Firebird Conference 2019

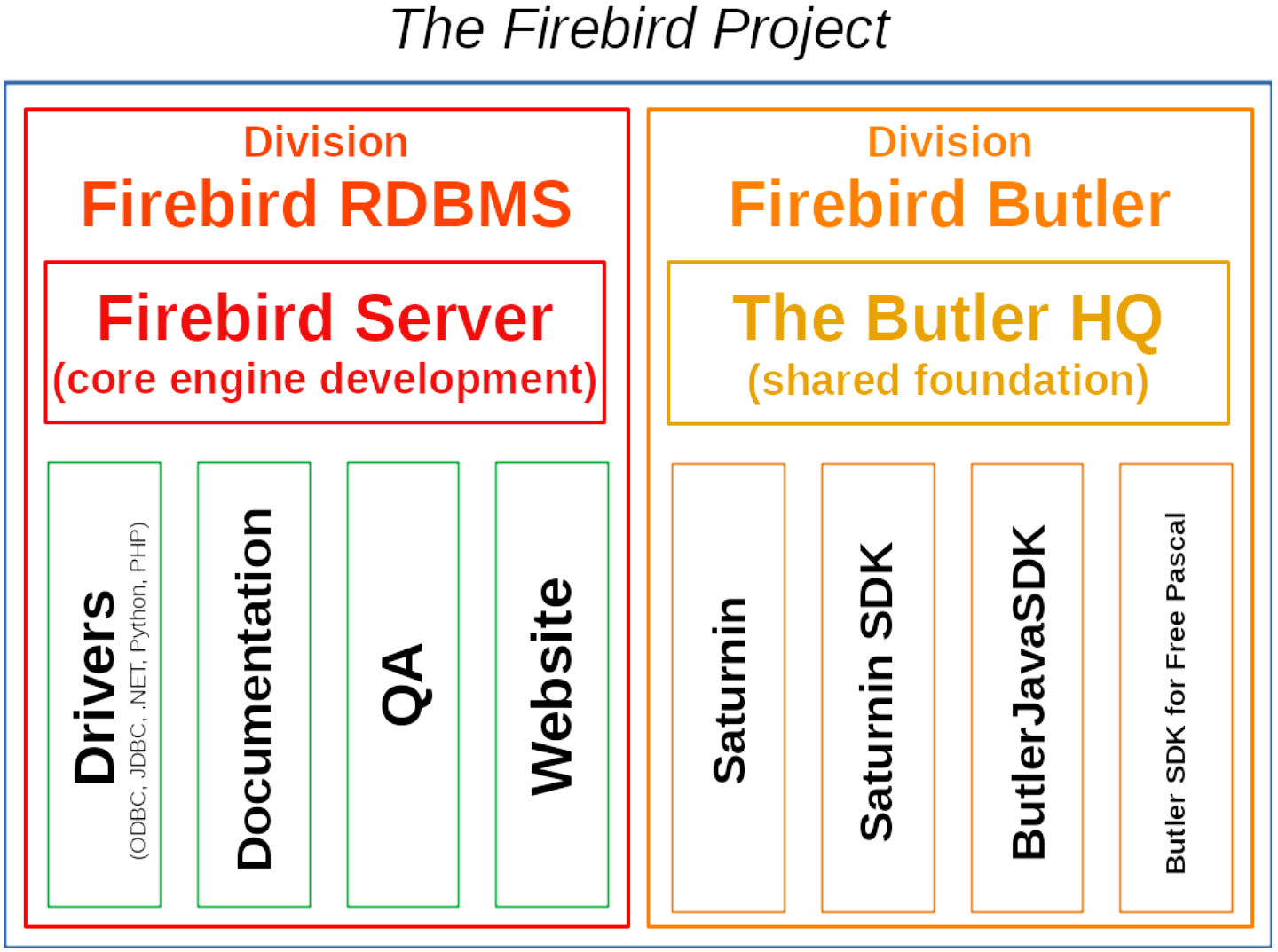
Berlin, 17-19 October

IBSurgeon

Project structure 2000~2018



Project structure 2019~



Why new division?

It’s part of a strategic plan to address the pressing problems of the project and Firebird users

Part I.
The Pressing Problems

Problem No. 1


2000 – Establishment
2001
2002 – 1.0.0
2003
2004 – 1.5.0
2005
2006 – 2.0.0
Small upgrade barrier
2007
2008 – 2.1.0
2009
2010 – 2.5.0
2011
2012
2013
2014
2015
2016 – 3.0.0
Great upgrade barrier
2017
2018 – 3.0.4
“Real” 3.0
2019
2020 – 4.0.0
End of v2.5

The “Great Divide” problem

- ~80% of current v2.5 server users **will not** upgrade their production systems to v3.0 in next two years, or at all
  - After 12 years, users have great investments in working 2.x-based systems
  - Upgrade costs often outweigh the benefits
- v3.0 is primarily deployed to **new** projects
- It will take many years for v3.0 deployments to exceed v2.5, unless Firebird will be adopted by many new users for major projects

Problem No. 1
Project perspective – part I.

Demand for new versions of Firebird

from current users

has decreased

Problem No. 1
Project perspective – part II.

We are failing to get

brand new users

deploying Firebird to major projects

(due to low "cool" factor)

Problem No. 1
User perspective

While Firebird ***engine*** is still perceived as very good and dependable, there is a growing ***feeling of uncertain future with Firebird Project***:

- ✔ No vision
- ✔ Low shock resistance
- ✔ No velocity
- ✔ Aging
- ✔ Technologically lagging

A taint among current users, greater impact on potential new users

Part II.
The Strategic Plan

Premises

1. Without a clear vision there is no direction
2. Without direction, action is not meaningful
3. Velocity matters
4. The “Great Divide” will not disappear anytime soon, so we have to fight on two fronts
5. A small and aging team is both a constraint and a problem to be solved

Strategy of fighting on two fronts I.

Nothing can be done for users facing the transition gap more than making the transition more rewarding

- Address common user problems we neglected so far
  - ✔ data storage management, problem detection, self-tuning
  - ✔ simplifying and speeding up routine administration activities
- Tune version 5 to perfection, making it a new baseline
  - ✔ focus on enhancements, simplifications and efficiency instead of new features
  - ✔ nobody wants another upgrade barrier anytime soon
  - ✔ users prefer gradual improvements with easy and safe roll-out updates (no big leaps that destabilize critical deployments)

Strategy of fighting on two fronts II.

Where it makes sense:

- Attract new users by innovation rather than copying others
  - We are small, it's better to occupy our own unique niche than fighting for scraps after big ones in mainstream
  - Also, brand new things bring brand new users
- Not to be afraid of creative solutions to old problems
  - difference is an important factor in deciding between solutions to the same problem
  - creativity attracts creative and passionate people

It is better to try to be a leader than a follower

Premises - 2nd iteration

- At least for near future, the solution is not in development of big new features in the Firebird engine.
  - ✔ Engine development needs time to consolidate the achieved, and not rush to the next big thing
  - ✔ Most problems that require new features could be solved in other layers with minimal requirements to core engine development
- Due to limited resources, it is adequate to try combine the solution to common user problems (first front) with innovations and new products (second front)
- Chosen strategy inevitably leads to expansion into adjacent territory dominated by Firebird support and tool creation companies

Tactics

Do Epic Shit

or die trying

Part III.

The Origin of Firebird Butler Idea

Goal reminder

1) We need innovative and creative new product(s) that solve common problems of Firebird users
2) We need to cooperate with companies affiliated to Firebird Project that provide support and tools

The best way is to join forces on shared interests!

The Origin of Firebird Butler Idea

The idea of Firebird Butler arose in mid-2018 from a discussion about shared pressing problems between two Firebird Project administrators, a member of the Firebird Foundation committee, the president of the Firebird Foundation, and two IBPhoenix engineers

At that time, IBPhoenix was developing a new internal tooling system, which, thanks to its focus and properties, fits perfectly into the strategic plan

Therefore, it was decided to modify the project concept, scope and development methods and restart it as an open project within or close by the Firebird Project

IBPhoenix's experience, part I.

- In fact, all users have only one problem: *ensuring stable operation of Firebird within the required parameters*
- There are practically no two users who need to solve this problem under the same conditions and with the same parameters

IBPhoenix’s experience, part II.

Providing user support is like cooking

Mostly you cook only a few different meals, but almost always in a foreign kitchen, in different quantities, for different tastes, and from the ingredients they bring you

You can cook good food in borrowed pots, but never without your own knives

You can't buy the best food at the store. You have to cook it yourself or have it prepared by a chef

IBPhoenix's experience, part III.

- TANSTAASSOS
  There Ain't No Such Thing As A 100% Satisfactory Solution On Sale
  You always end up with more or less your own custom solution
- Good tools designed for engineers and not users are scarce
- Many tools are the victims of the evolution towards extending functionality to complex versatility
  - ✔ They do many things but few of them do well
  - ✔ They are full of "leaky abstractions[tm]"
- Many tools are intended only to support bad temporary solutions that have become permanent

IBPhoenix's approach

- Develop engineering tools to build ***efficient***, ***robust*** and ***scalable*** solutions to ensure stable operation of Firebird within the required parameters
- Then create a set of templates to solve the most common situations
- Finally, create a collection of pre-configured distribution packages for deployment in everyday situations

The foundation of new internal tooling system

- Imagine a system for creating complex solutions by linking small program components
  eg Delphi Non-Visual Components – but connected by sending messages instead direct procedure calls
- Now add the possibility that the existing system can be adapted to new requirements at any time by changing / adding / removing its components
- And that components can run together in one process or distributed in different ways over a network
- And it runs on many platforms and supports many programming languages

So far nothing so revolutionary

The innovative difference

- Has the scalability, reliability and flexibility of distributed messaging-based systems

  but at the same time with the same code

- It has very small footprint and is highly efficient in compact deployment (like single-user applications)

from embedded to enterprise^{tm}

***** one code to rule them all *****

Too beautiful to be true

- There was a time, when it was not imaginable that Pascal IDE (ultrafast compiler + editor + application runtime - with floating point numerics) can fit into 16Kb
- There was a time, when idea that transactions may not require locks was insane
- Now is a time, when nobody can imagine distributed components without full HTTP(S) or AMQP stack, REST, JSON, XML and “containers” (to name a few “absolute necessities”), that are natural killers for “personal” and “embedded”
- It’s also believed that these toys are only for the “***big boys***”

The ZeroMQ, part I.

- It's an elegant, small yet powerful ***library*** that ***handles the messaging fundamentals*** while allowing users to build various messaging architectures
- The interface is based on the Berkeley sockets API, with ***additional*** functionality
- It provides ***services*** and ***patterns*** to do brokered or broker-less systems, sync and async messaging, various request/reply conversations, push/pull pipes, publisher/subscriber multicasts, workload pipelines, authentication, CURVE security, and various protocols (tcp, ipc, inproc, pgm/epgm, vmci)
- It significantly simplifies the development of viable custom systems with load balancing, high availability or reliability features, dynamic mashups, cloud architectures etc. It does not do these things for you, but you can do it yourself reliably without too much effort, ***in a way and to the extent that fits your needs***

The ZeroMQ, part II.

- As a cherry on top, it helps with reliable multi-thread applications using ***inproc*** sockets for fast and safe pipelines, or multi-process ones with ***ipc*** sockets
- Of course it's not perfect. It has limits, issues and dark corners like any piece of software out there, but we think that it provides everything we would need at a level that is good enough
- Comes with the low-level C API. High-level bindings exist in ***40+*** languages including ***Python***, ***Delphi***, ***FreePascal***, ***Java***, ***PHP***, ***Ruby***, ***C***, ***C++***, ***C#***, ***Erlang***, ***Perl*** and more
- It also comes in a ***pure Java stack*** called ***JeroMQ***, and a ***pure C# stack*** called ***NetMQ***. These are both official projects supported by the ZeroMQ community
- Used at AT&T, Cisco, EA, Los Alamos Labs, NASA, Weta Digital, Zynga, Spotify, Samsung Electronics, IBM, Microsoft, CERN etc.

Part IV.
The Firebird Butler

The Butler layers

1) ***The Butler Development Platform***
   engineering tools to build ***efficient***, ***robust*** and ***scalable*** solutions

2) ***The Butler Services***
   set of building blocks to build solutions to ensure stable operation of Firebird within the required parameters

3) ***The Firebird Butler*** (as end user product)
   collection of templates to solve the most common situations and pre-configured distribution packages for deployment in everyday situations

Openness to the bone

- ***Open specifications*** defined using *Consensus-Oriented Specification System*
  - ✔ *define basic features of Butler Services*
  - ✔ *messaging protocols*
  - ✔ *anything else that helps to set the common ground for involved parties*
- ***The Firebird project is a governing body***, but anyone can join and participate in the way and to the extent that it deems appropriate
- The specifications do not describe how platform should be implemented, neither how individual services should do their jobs

The Butler Development Platform

- Platform ***specifications*** *(a set of **RFC** documents)* are blueprints for Platform ***implementation*** in various programming languages
- The ***reference implementation*** is in the ***Python*** programming language (codename ***Saturnin SDK***)
- The ***ButlerJavaSDK*** and ***Butler SDK for Free Pascal*** subprojects have been opened to implement the platform in these programming languages
- Because a platform specification does not define exact API or implementation details, the individual implementations may vary in their design, architecture, features and API provided for the ***service developers***

Butler Services, part I.

- A Butler Service is basically a piece of software that uses ***ZeroMQ socket*** and ***Firebird Butler Service Protocol*** (***FBSP***) for communication over this ZeroMQ channel
- A service could use multiple ZeroMQ sockets for various purposes, but only one primary socket is required to support the Butler Service protocol
- They ***could do anything***, but a well designed service does only one task, or a small set of closely related tasks within single category
- We also introduced a category of ***microservices*** that do not communicate via *FBSP* but use ***some other standard protocol*** to communicate with other components *(eg Firebird Butler Data Pipe Protocol - **FBDP**)*

Butler Services, part II.

While respecting the rule of simplicity, services can be divided into several basic types:

- ***Measuring*** services, that collect and pass on data
  For example it may collect data from monitoring tables
- ***Processing*** services, that take data on input, do something (on them) and have some data on output
  This general category includes services that perform analytics, data transformation, brokers, bridges, routers, aggregators, load balancers etc.
- ***Provider*** services, that do things on request
  For example perform a database backup
- ***Control*** services, that manage other services

Where we want to go
Planned services: *Measuring services*

At first we want to take advantage of *existing* Firebird features *(API, monitoring tables, trace service, user queries etc.)* to provide data about transactions, connections, queries, security, configuration, resources, availability, operational information (logs, gstat output etc.) ***in more consolidated format***

Consequently, we assume that additional user requirements for ***new information sources*** or ***formats*** will be met in collaboration with the Firebird engine developers

Where we want to go
Planned services: *Processing services*

We plan to create at least the following set of services in this category:

- ***Notification*** service to send reports and alerts via e-mail and other means
- ***Logging*** service, to collect log entries from other services
- User-defined state machine ***analytics*** on measured data ***to trigger actions*** or ***emit signals***
- Export of metrics to the ***Graphite*** open source monitoring product (primarily as metrics storage and a visualization solution)
- Import connector for the ***python-diamond*** open source daemon that ***collects system metrics*** such as cpu, memory, network, i/o, load, disk etc.

Where we want to go
Planned services: *Provider services*

We plan to create at least the following set of services in this category:

- ***Registry*** service for services and other resources, as a solution for central configuration needs
- Services for ***Firebird-related tasks*** like: backup & restore (gbak and nbackup), sweep and other tasks provided by gfix, cancellation of connections and transactions, user defined SQL commands and scripts
- ***Scheduler*** service to run jobs on a regular basis
- Service to ***run user defined action*** -> external program or script
- Extensible ***storage*** service *(it will provide at least file-based storage)*

Where we want to go
Planned services: *Control services*

We plan to create at least the following set of services in this category:

- General ***control*** service that uses Butler protocol specifications to keep track of the ***state*** and ***configuration*** of ***registered running services***, could emit signals and other information messages about it, and provide ***management*** service activities within the scope defined by Butler specifications
- A ***command-line tool*** to interact with this control service

Current state, part I.

Firebird Butler Specifications, part I.

Life cycle of specification

Raw → Draft → Stable → Deprecated → Retired

- Stable
  - ✔ ***1/C4*** *- Collective Code Construction Contract*
  - ✔ **2/COSS** *- Consensus-Oriented Specification System*
- Draft
  - ✔ ***3/FBSD*** *- Firebird Butler Service Definition*
    reference implementation: Saturnin+Saturnin SDK
  - ✔ ***4/FBSP*** *- Firebird Butler Service Protocol*
    reference implementation: Saturnin SDK
  - ✔ ***9/FBDP*** *- Firebird Butler Data Pipe Protocol*
    reference implementation: Saturnin SDK

Current state, part II.

Firebird Butler Specifications, part II.

- Raw *(stubs)*
  - ***5/FBLP*** *- Firebird Butler Log Protocol*
  - ***6/SSTP*** *- Service State Protocol*
  - ***7/RSCFG*** *- Remote Service Configuration Protocol*
  - ***8/RSCTRL*** *- Remote Service Control Protocol*

Current state, part III.

Saturnin SDK

Early Beta

- ✔ ready for early testing
- ✔ implementation of ***FBSD***, ***FBSP***, ***FBDP***
- ✔ only code, no documentation yet (**will be soon**)
- ✔ ***example services*** and ***microservices***
- ✔ ***raw*** service execution and testing enviroment

Introductory presentation on the 3rd day

Current state, part IV.

Saturnin

Alpha (working prototypes)

- ✔ prototype of ***saturnin-node*** service
- ✔ prototype of saturnin CLI ***console***
- ✔ prototype of ***firebird-log*** microservice

Introductory presentation on the 3rd day

Part V.
Future outlook

Future outlook

- The *early* experience with Butler Development Platform gave very promising results
- At the current speed we expect ***first*** tangible results at 3rd layer (templates & preconfigured packages) by the end of next year
- There is still a long way to go, but as the technology used has a ***wide range of uses*** and ***great potential***, ***we hope*** that ***new collaborators*** will ***join*** us soon
- In such a case, we can get a big acceleration and possibly start the snowball effect

New chapter of Firebird project

Although we have begun this ambitious project to move the Firebird project forward, it opens up the possibility of bringing the **Firebird™ brand** *to areas that have not yet been anticipated*

Questions?

Thanks for your attention

Contacts:

- Email: pcisar@ibphoenix.cz
- www.ibphoenix.com

Firebird Butler:

- git: https://github.com/FirebirdSQL/Butler
- Documentation: http://firebird-butler.rtfd.io/
- Mailing list: https://groups.google.com/d/forum/firebird-butler