Saftlib v2.0

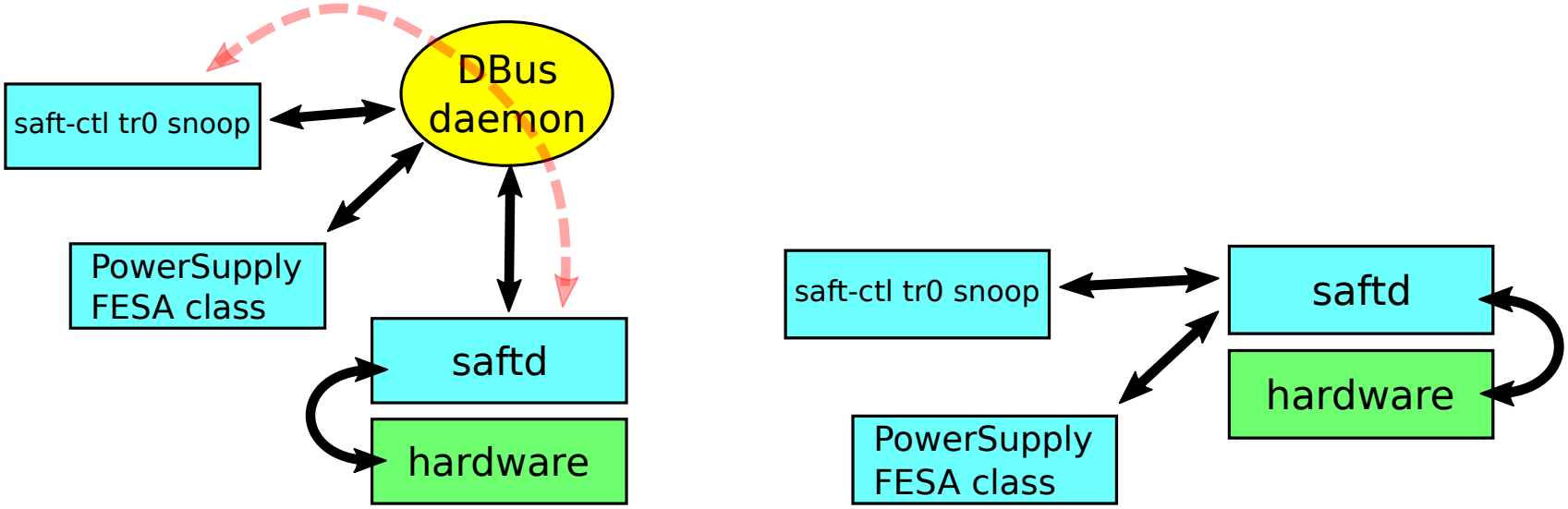
Outline

- API
  - No DBus anymore
    - Type replacements
    - Signal driven iteration
  - UTC support with saftlib::Time
    - Read time and inject events
    - Callback function
    - Connect callback to signal
- Command line tools
  - UTC support in saftlib command line tools

No DBus anymore

Advantages

- No dbus-daemon process needed (saftbus runs inside saftd)
- Fixed number of slots for client processes. Are 32 enough?
- Faster than DBus, but potentially more fragile (Signals must be processed).
- Glibmm not used anymore
  - API change required (mostly search & replace)



Type replacements

Saftlib 1.4 (C++11 optional)

```
glib::RefPtr<T>
Glib::init()
Gio::init()
Glib::ustring
g(u)int
g(u)int8
g(u)int16
g(u)int32
g(u)int64
Glib::MainContext::get_default()
                  ->iteration(true)
catch(Glib::Error &e)
```

Saftlib 2.0 (C++11 required)

```
std::shared_ptr<T>
// nothing
// nothing
std::string
(unsigned) int
(u)int8_t
(u)int16_t
(u)int32_t
(u)int64_t
saftlib::wait_for_signal()

catch(saftbus::Error &e)
```

Signal driven iteration

Saftlib 1.4

- ```
  Glib::RefPtr<Glib::MainLoop> loop = Glib::MainLoop::create();
  loop->run();
  ```
- ```
  Glib::RefPtr<Glib::MainLoop> loop = Glib::MainLoop::create();
  while(true) {
    loop->get_context()->iteration(true);
  }
  ```
- ```
  while(true) {
    Glib::MainContext::get_default()->iteration(true);
  }
  ```

Saftlib 2.0

```
while(true) {
  saftlib::wait_for_signal();
}
```
```

UTC support (this is a CCT requirement)

CCT requests

- Control system has a single time base (UTC)
  (Leap second $\Rightarrow$ technical stop)
- UTC support in the Saftlib API
- Deprecate old Saftlib API and remove it after transition period

Inevitable consequences

- User code needs to be changed [1]
- Saftlib API has to be extended/changed

Design goal

- If it compiles, it should work correctly

[1] Even if Saftlib would keep the existing interface with UTC instead of TAI. No compiler support possible in this case!

Example: Read time and inject timing event (Saftlib 1.4)

Saftlib 1.4

```
guint64 TAItime = receiver->ReadCurrentTime();
TAItime += 1000000000; // next event one second later
receiver->InjectEvent(id,param,TAItime);
```

You want UTC?

Add offset yourself!

InjectEvent xml interface definition

New type "T" ⇒ saftlib::Time

```
<interface name="de.gsi.saftlib.TimingReceiver">
  ...
  <method name="InjectEvent">
    <arg direction="in" type="t" name="event"/>
    <arg direction="in" type="t" name="param"/>
    <arg direction="in" type="T" name="time"/>
  </method>
  <method name="InjectEvent" deprecated="yes">
    <arg direction="in" type="t" name="event"/>
    <arg direction="in" type="t" name="param"/>
    <arg direction="in" type="t" name="time"/>
  </method>
  ...
</interface>
```

Example: Read time and inject timing event (Saftlib 2.0)

Saftlib 2.0 - no major differences when interacting with saftlib

```
saftlib::Time time = receiver->CurrentTime(); // name changed
time += 1000000000; // next event: one second later
receiver->InjectEvent(id,param,time);
```

Example: Read time and inject timing event (Saftlib 2.0)

Saftlib 2.0 - no major differences when interacting with saftlib

```
saftlib::Time time = receiver->CurrentTime(); // name changed
time += 1000000000; // next event: one second later
receiver->InjectEvent(id,param,time);
```

You want UTC?

```
uint64_t TAItime = time.getTAI();
uint64_t UTCtime = time.getUTC();
int64_t  UTCoffs = time.getUTCOffset(); // TAI-UTC
bool leap        = time.isLeapUTC();
```

Example: Read time and inject timing event (Saftlib 2.0)

Saftlib 2.0 - no major differences when interacting with saftlib

```
saftlib::Time time = receiver->CurrentTime(); // name changed
time += 1000000000; // next event: one second later
receiver->InjectEvent(id,param,time);
```

You want UTC?

```
uint64_t TAItime = time.getTAI();
uint64_t UTCtime = time.getUTC();
int64_t  UTCoffs = time.getUTCOffset(); // TAI-UTC
bool leap        = time.isLeapUTC();

saftlib::Time ambiguous = 12345;                          // Error!
saftlib::Time my_time_1 = saftlib::makeTimeTAI(12345); // ok
saftlib::Time my_time_2 = saftlib::makeTimeUTC(12345); // ok
saftlib::Time my_time_3 = saftlib::makeTimeUTC(12345,true); // ok
```

Callback function

Saftlib 1.4

```
void on_action_1_4(
      const guint64 & event,
      const guint64 & param,
      const guint64 & deadline,
      const guint64 & executed,
      const guint16 & flags)
{
  // using absolute time value
  //   outside saftlib
  cout << deadline << endl;


  // use absolute time value
  //   within saftlib API
  receiver->InjectEvent(
               event,
               param,
               deadline+1000000000)
  //...
}
```

Saftlib 2.0

```
void on_action_2_0(
      const uint64_t &      event,
      const uint64_t &      param,
      const saftlib::Time & deadline,
      const saftlib::Time & executed,
      const uint16_t &      flags)
{
  // change required!
  //   choose TAI or UTC
  cout << deadline.getTAI() << endl;
  cout << deadline.getUTC() << endl;

  // no change needed!
  // saftlib accepts saftlib::Time
  receiver->InjectEvent(
               event,
               param,
               deadline+1000000000)
  // ...
}
```

Connect callbacks to signals

Saftlib 1.4

```
Glib::ustring object_path("/de/gsi/saftlib/tr0/software/_2/_1303455736");
Glib::RefPtr<saftlib::SoftwareActionSink_Proxy> proxy =
         saftlib::SoftwareActionSink_Proxy::create(object_path);

proxy->Action.connect(sigc::ptr_fun(&on_action_1_4)); // callback
```

Saftlib 2.0

```
std::string object_path("/de/gsi/saftlib/tr0/software/_2/_1303455736");
std::shared_ptr<saftlib::SoftwareActionSink_Proxy> proxy =
         saftlib::SoftwareActionSink_Proxy::create(object_path);

proxy->Action.connect(sigc::ptr_fun(&on_action_1_4)); // works, but is
                                                      //   deprecated!
proxy->SigAction.connect(sigc::ptr_fun(&on_action_2_0));
```

Questions to the users

Consequences of API change

- Old API still works but is marked as deprecated (compiler emits warnings)
- UTC related changes are not fixable with "search & replace"
  Strategy for the transition to the new API:
  If a 64-bit integer is used as absolute time: Replace the type with saftlib::Time and fix all following compiler errors.
- Some methods/signals have different names in new API, e.g.:
  - `guint64 ReadCurrentTime()` ⇒ `saftlib::Time CurrentTime()`
  - `Action(..., guint64)` ⇒ `SigAction(..., saftlib::Time)`

Questions

- Suggestions/Improvements?
- Lifetime of deprecated API?

UTC support in saftlib command line tools

Additional option -U

Absolute time values are written as UTC, leap seconds have a '*'

- saft-ctl
- saft-io-ctl
- saft-fg-ctl
- saft-pps-gen

Additional option -L only together with -U and saft-ctl inject

If the injected UTC second is ambiguous:

- -U means: the earlier possibility is used
- -UL means: the later possibility is used

leap second transition in saft-ctl

saft-ctl tr0 -vU snoop 0 0 0 (fake leap second was inserted)

```
tDeadline: 2019-05-20 13:25:23.329989936  FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:23.641389944  FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:23.956582992  FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:24.308865688  FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:24.574189744  FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:24.882593872  FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:25.197488224  FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:25.502896712  FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:25.862068832  FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:25.127592232* FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:25.444635584* FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:25.753937671* FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:26.070628328  FID: 0 GID: ....
tDeadline: 2019-05-20 13:25:26.428551344  FID: 0 GID: ....
```

Further reading

`https://www-acc.gsi.de/wiki/Timing/Saftlib2MigrationGuide`

- full list of API changes
- more details and explanations
- larger examples